

# Namespace PAC

## Classes

[Finder](#)

[Preferences](#)

[Preferences.Preference<T>](#)

## Enums

[FlipAxis](#)

[RotationAngle](#)

This should be interpreted as being clockwise.

# Class Finder

Namespace: [PAC](#)

```
public class Finder
```

## Inheritance

[object](#) ← Finder

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

# Properties

## animationManager

```
public static AnimationManager animationManager { get; }
```

Property Value

[AnimationManager](#)

## clipboard

```
public static Clipboard clipboard { get; }
```

Property Value

[Clipboard](#)

## colourPicker

```
public static GlobalColourPicker colourPicker { get; }
```

Property Value

[GlobalColourPicker](#)

## dialogBoxManager

```
public static DialogBoxManager dialogBoxManager { get; }
```

Property Value

[DialogBoxManager](#)

## drawingArea

```
public static DrawingArea drawingArea { get; }
```

Property Value

[DrawingArea](#)

## fileManager

```
public static FileManager fileManager { get; }
```

Property Value

[FileManager](#)

## fileTabsManager

```
public static FileTabsManager fileTabsManager { get; }
```

Property Value

[FileTabsManager](#)

## gridManager

```
public static GridManager gridManager { get; }
```

Property Value

[GridManager](#)

## imageEditManager

```
public static ImageEditManager imageEditManager { get; }
```

Property Value

[ImageEditManager](#)

## inputSystem

```
public static InputSystem inputSystem { get; }
```

Property Value

[InputSystem](#)

## keyboard

```
public static Keyboard keyboard { get; }
```

Property Value

[Keyboard](#)

## layerManager

```
public static LayerManager layerManager { get; }
```

Property Value

[LayerManager](#)

## mouse

```
public static Mouse mouse { get; }
```

Property Value

[Mouse](#)

## themeManager

```
public static ThemeManager themeManager { get; }
```

Property Value

[ThemeManager](#)

## tileOutlineManager

```
public static TileOutlineManager tileOutlineManager { get; }
```

Property Value

[TileOutlineManager](#)

## tilesetManager

```
public static TilesetManager tilesetManager { get; }
```

Property Value

[TilesetManager](#)

## toolbar

```
public static Toolbar toolbar { get; }
```

Property Value

[Toolbar](#)

## uiManager

```
public static UIManager uiManager { get; }
```

Property Value

[UIManager](#)

## undoRedoManager

```
public static UndoRedoManager undoRedoManager { get; }
```

Property Value

[UndoRedoManager](#)

# Enum FlipAxis

Namespace: [PAC](#)

```
public enum FlipAxis
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Horizontal = 2

Minus45Degrees = 4

None = 0

Vertical = 1

\_45Degrees = 3

# Class Preferences

Namespace: [PAC](#)

```
public static class Preferences
```

## Inheritance

[object](#) ← Preferences

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Fields

### startupAnimationFramerate

```
public static readonly Preferences.Preference<int> startupAnimationFramerate
```

#### Field Value

[Preferences.Preference<int>](#)

### startupBrushSize

```
public static readonly Preferences.Preference<int> startupBrushSize
```

#### Field Value

[Preferences.Preference<int>](#)

### startupFileHeight

```
public static readonly Preferences.Preference<int> startupFileHeight
```

Field Value

[Preferences.Preference<int>](#)

## startupFileWidth

```
public static readonly Preferences.Preference<int> startupFileWidth
```

Field Value

[Preferences.Preference<int>](#)

## startupPrimaryColour

```
public static readonly Preferences.Preference<Color> startupPrimaryColour
```

Field Value

[Preferences.Preference<Color>](#)

## startupSecondaryColour

```
public static readonly Preferences.Preference<Color> startupSecondaryColour
```

Field Value

[Preferences.Preference<Color>](#)

## transparentCheckerboardColour1

This is the colour that will be in the bottom left / top right of the checkerboard.

```
public static readonly Preferences.Preference<Color32>
transparentCheckerboardColour1
```

## Field Value

[Preferences.Preference<Color32>](#)

## transparentCheckerboardColour2

This is the colour that will be in the bottom right / top left of the checkerboard.

```
public static readonly Preferences.Preference<Color32>
transparentCheckerboardColour2
```

## Field Value

[Preferences.Preference<Color32>](#)

## Methods

### CreatePreference(string, string, Color)

```
public static Preferences.Preference<Color> CreatePreference(string displayName,
string playerPrefsKey, Color defaultValue)
```

## Parameters

displayName [string](#)

playerPrefsKey [string](#)

defaultValue Color

## Returns

[Preferences.Preference<Color>](#)

## CreatePreference(string, string, Color32)

```
public static Preferences.Preference<Color32> CreatePreference(string displayName,  
string playerPrefsKey, Color32 defaultValue)
```

### Parameters

displayName [string](#)

playerPrefsKey [string](#)

defaultValue Color32

### Returns

[Preferences.Preference<Color32>](#)

## CreatePreference(string, string, int, int, int)

```
public static Preferences.Preference<int> CreatePreference(string displayName,  
string playerPrefsKey, int defaultValue, int minValue = -2147483648, int maxValue  
= 2147483647)
```

### Parameters

displayName [string](#)

playerPrefsKey [string](#)

defaultValue [int](#)

minValue [int](#)

maxValue [int](#)

### Returns

[Preferences.Preference<int>](#)

# Class Preferences.Preference<T>

Namespace: [PAC](#)

```
public class Preferences.Preference<T>
```

## Type Parameters

T

### Inheritance

[object](#) ← Preferences.Preference<T>

### Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

### Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

Preference(string, Func<T>, Action<T>)

```
public Preference(string displayName, Func<T> getter, Action<T> setter)
```

## Parameters

displayName [string](#)

getter [Func](#)<T>

setter [Action](#)<T>

## Properties

# displayName

```
public string displayName { get; }
```

Property Value

[string ↗](#)

## Methods

### Get()

Gets the value of the preference. Can also be done via casting.

```
public T Get()
```

Returns

T

### Set(T)

Change the value of the preference and save it.

```
public void Set(T value)
```

Parameters

**value** T

## Operators

### implicit operator T(Preference<T>)

Gets the value of the preference.

```
public static implicit operator T(Preferences.Preference<T> pref)
```

Parameters

pref [Preferences.Preference<T>](#)

Returns

T

# Enum RotationAngle

Namespace: [PAC](#)

This should be interpreted as being clockwise.

```
public enum RotationAngle
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Minus90 = -90

\_0 = 0

\_180 = 180

\_90 = 90

# Namespace PAC.Animation

## Classes

### [AnimationKeyFrame](#)

A class representing a single keyframe for a layer.

### [AnimationKeyFrame.JsonConverter](#)

### [AnimationManager](#)

Handles the animation timeline and playback of animations. The animation system is still a work in progress.

### [FrameNotch](#)

A class for the frame number markers on the animation timeline.

### [KeyFrameIcon](#)

Stores the data about a keyframe icon on the animation timeline.

# Class AnimationKeyFrame

Namespace: [PAC.Animation](#)

A class representing a single keyframe for a layer.

```
public class AnimationKeyFrame
```

## Inheritance

[object](#) ← AnimationKeyFrame

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### AnimationKeyFrame(AnimationKeyFrame)

Creates a deep copy of the AnimationKeyFrame.

```
public AnimationKeyFrame(AnimationKeyFrame animationKeyFrame)
```

## Parameters

animationKeyFrame [AnimationKeyFrame](#)

### AnimationKeyFrame(int, Texture2D)

```
public AnimationKeyFrame(int frame, Texture2D texture)
```

## Parameters

**frame** [int](#)

**texture** Texture2D

## Fields

### frame

The number of the frame this keyframe is on.

```
public int frame
```

### Field Value

[int](#)

### texture

The texture displayed at this keyframe.

```
public Texture2D texture
```

### Field Value

Texture2D

## Methods

### DeepCopy()

Creates a deep copy of the AnimationKeyFrame.

```
public AnimationKeyFrame DeepCopy()
```

Returns

[AnimationKeyFrame](#)

# Class AnimationKeyFrame.JsonConverter

Namespace: [PAC.Animation](#)

```
public class AnimationKeyFrame.JsonConverter :  
JsonConversion.JsonConverter<AnimationKeyFrame, JsonData.Object>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<AnimationKeyFrame, JsonData.Object>](#) ←  
AnimationKeyFrame.JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<AnimationKeyFrame,  
JsonData.Object>.ToJson\(AnimationKeyFrame\)](#) ,  
[JsonConversion.JsonConverter<AnimationKeyFrame,  
JsonData.Object>.FromJson\(JsonData.Object\)](#) ,  
[JsonConversion.JsonConverter<AnimationKeyFrame, JsonData.Object>.FromJson\(JsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### JsonConverter(SemanticVersion)

```
public JsonConverter(SemanticVersion fromJsonFileFormatVersion)
```

## Parameters

fromJsonFileFormatVersion [SemanticVersion](#)

## Methods

## FromJson(Object)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override AnimationKeyFrame FromJson(JsonData.Object jsonData)
```

Parameters

jsonData [JsonData.Object](#)

Returns

[AnimationKeyFrame](#)

## ToJson(AnimationKeyFrame)

Attempts to convert the C# object into JSON data.

```
public override JsonData.Object ToJson(AnimationKeyFrame keyFrame)
```

Parameters

keyFrame [AnimationKeyFrame](#)

Returns

[JsonData.Object](#)

# Class AnimationManager

Namespace: [PAC.Animation](#)

Handles the animation timeline and playback of animations. The animation system is still a work in progress.

```
public class AnimationManager : MonoBehaviour
```

## Inheritance

[object](#) ← AnimationManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### framerate

```
public int framerate
```

Field Value

[int](#)

### onionSkinColour

```
public Color onionSkinColour
```

Field Value

Color

# showOnionSkin

```
public bool showOnionSkin
```

Field Value

[bool](#)

## Properties

### currentFrameIndex

The number of the frame currently being displayed.

```
public int currentFrameIndex { get; set; }
```

Property Value

[int](#)

## Methods

### AddKeyFrame()

Extends the length of the animation by adding one frame to the end.

```
public void AddKeyFrame()
```

### DebugLogKeyFrames()

For debugging purposes. Prints the current frame number and the frame numbers of each keyframe.

```
public void DebugLogKeyFrames()
```

## DeleteSelectedKeyFrame()

Deletes the selected key frame.

```
public void DeleteSelectedKeyFrame()
```

## Pause()

Pauses the animation playback.

```
public void Pause()
```

## Play()

Starts/resumes the animation playback.

```
public void Play()
```

## RemoveKeyFrame()

Reduces the length of the animation by removing the final frame.

```
public void RemoveKeyFrame()
```

## Stop()

Stops the animation playback and resets the current frame number back to the start.

```
public void Stop()
```

## SubscribeToOnCurrentFrameIndexChanged(UnityAction)

Event is invoked when the current frame number changes.

```
public void SubscribeToOnCurrentFrameIndexChange(UnityAction call)
```

Parameters

**call** UnityAction

## SubscribeToOnKeyFrameAdded(UnityAction)

Event is invoked when a keyframe is added.

```
public void SubscribeToOnKeyFrameAdded(UnityAction call)
```

Parameters

**call** UnityAction

## SubscribeToOnKeyFrameDeleted(UnityAction)

Event is invoked when a keyframe is deleted.

```
public void SubscribeToOnKeyFrameDeleted(UnityAction call)
```

Parameters

**call** UnityAction

# Class FrameNotch

Namespace: [PAC.Animation](#)

A class for the frame number markers on the animation timeline.

```
public class FrameNotch : MonoBehaviour
```

## Inheritance

[object](#) ← FrameNotch

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### frameNum

```
public int frameNum { get; set; }
```

Property Value

[int](#)

## Methods

### SetFrameNumber(int)

```
public void SetFrameNumber(int num)
```

Parameters

num [int](#)

# Class KeyFrameIcon

Namespace: [PAC.Animation](#)

Stores the data about a keyframe icon on the animation timeline.

```
public class KeyFrameIcon : MonoBehaviour
```

## Inheritance

[object](#) ↗ KeyFrameIcon

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Fields

### frameIndex

The frame number this keyframe is on.

```
public int frameIndex
```

Field Value

[int](#) ↗

### layerIndex

The layer index this keyframe is on.

```
public int layerIndex
```

Field Value

[int](#) ↗

# Namespace PAC.Clipboard

## Classes

### [Clipboard](#)

A class to handle copy/paste functionality. Not yet properly implemented.

# Class Clipboard

Namespace: [PAC.Clipboard](#)

A class to handle copy/paste functionality. Not yet properly implemented.

```
public class Clipboard : MonoBehaviour
```

## Inheritance

[object](#) ↪ Clipboard

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Methods

### Copy()

```
public void Copy()
```

### Cut()

```
public void Cut()
```

### Paste()

```
public void Paste()
```

# Namespace PAC.Colour

## Classes

### [BlendMode](#)

A class for blend modes.

### [BlendMode.JsonConverter](#)

### [ColorExtensions](#)

## Structs

### [HSL](#)

Represents a colour in HSL form.

### [HSV](#)

Represents a colour in HSV form.

# Class BlendMode

Namespace: [PAC.Colour](#)

A class for blend modes.

```
public class BlendMode
```

## Inheritance

[object](#) ← BlendMode

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

# Fields

## Add

Add blend mode.

```
public static readonly BlendMode Add
```

## Field Value

[BlendMode](#)

## Multiply

Multiply blend mode.

```
public static readonly BlendMode Multiply
```

Field Value

[BlendMode](#)

## Normal

Normal blend mode.

```
public static readonly BlendMode Normal
```

Field Value

[BlendMode](#)

## Overlay

Overlay blend mode.

```
public static readonly BlendMode Overlay
```

Field Value

[BlendMode](#)

## Replace

Replace blend mode.

```
public static readonly BlendMode Replace
```

Field Value

[BlendMode](#)

## Screen

Screen blend mode.

```
public static readonly BlendMode Screen
```

Field Value

[BlendMode](#)

## Subtract

Subtract blend mode.

```
public static readonly BlendMode Subtract
```

Field Value

[BlendMode](#)

## blendModes

All implemented blend modes.

```
public static readonly BlendMode[] blendModes
```

Field Value

[BlendMode\[\]](#)

## Methods

### Blend(Color, Color)

Blend the two colours using the blend mode's blend function.

```
public Color Blend(Color topColour, Color bottomColour)
```

Parameters

**topColour** Color

**bottomColour** Color

Returns

Color

## Equals(object)

Determines whether the specified object is equal to the current object.

```
public override bool Equals(object obj)
```

Parameters

**obj** [object](#)

The object to compare with the current object.

Returns

[bool](#)

[true](#) if the specified object is equal to the current object; otherwise, [false](#).

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## MultiplyColours(Color, Color)

Multiplies the colours component-wise.

```
public static Color MultiplyColours(Color colour1, Color colour2)
```

Parameters

colour1 Color

colour2 Color

Returns

Color

## StringToBlendMode(string)

Returns the blend mode with that name (case-insensitive).

```
public static BlendMode StringToBlendMode(string blendModeName)
```

Parameters

blendModeName [string](#)

Returns

[BlendMode](#)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

## [string](#)

A string that represents the current object.

# Operators

## operator ==(BlendMode, BlendMode)

```
public static bool operator ==(BlendMode a, BlendMode b)
```

### Parameters

a [BlendMode](#)

b [BlendMode](#)

### Returns

[bool](#)

## operator !=(BlendMode, BlendMode)

```
public static bool operator !=(BlendMode a, BlendMode b)
```

### Parameters

a [BlendMode](#)

b [BlendMode](#)

### Returns

[bool](#)

# Class BlendMode.JsonConverter

Namespace: [PAC.Colour](#)

```
public class BlendMode.JsonConverter : JsonConversion.JsonConverter<BlendMode,  
JsonData.String>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<BlendMode, jsonData.String>](#) ←  
BlendMode.JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<BlendMode, jsonData.String>.ToJson\(BlendMode\)](#) ,  
[JsonConversion.JsonConverter<BlendMode, jsonData.String>.FromJson\(jsonData.String\)](#) ,  
[JsonConversion.JsonConverter<BlendMode, jsonData.String>.FromJson\(jsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### FromJson(String)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override BlendMode FromJson(JsonData.String jsonData)
```

## Parameters

jsonData [JsonData.String](#)

Returns

[BlendMode](#)

## ToJson(BlendMode)

Attempts to convert the C# object into JSON data.

```
public override JsonData.String ToJson(BlendMode blendMode)
```

Parameters

blendMode [BlendMode](#)

Returns

[JsonData.String](#)

# Class ColorExtensions

Namespace: [PAC.Colour](#)

```
public static class ColorExtensions
```

## Inheritance

[object](#) ← ColorExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### Invert(Color, bool)

Inverts the colour, i.e. does 1 - value for each component.

```
public static Color Invert(this Color colour, bool invertAlpha = false)
```

#### Parameters

colour Color

invertAlpha [bool](#)

Whether to invert the alpha value as well.

#### Returns

Color

### ToHSL(Color)

```
public static HSL ToHSL(this Color color)
```

Parameters

**color** Color

Returns

[HSL](#)

## ToHSV(Color)

```
public static HSV ToHSV(this Color color)
```

Parameters

**color** Color

Returns

[HSV](#)

# Struct HSL

Namespace: [PAC.Colour](#)

Represents a colour in HSL form.

```
public struct HSL
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [object.Equals\(object, object\)](#) ,  
[object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### HSL(Color)

```
public HSL(Color rgb)
```

#### Parameters

rgb Color

### HSL(float, float, float)

```
public HSL(float hue, float saturation, float lightness)
```

#### Parameters

hue [float](#)

saturation [float](#)

`lightness` [float](#)

## HSL(float, float, float, float)

```
public HSL(float hue, float saturation, float lightness, float alpha)
```

### Parameters

`hue` [float](#)

`saturation` [float](#)

`lightness` [float](#)

`alpha` [float](#)

## Properties

### a

Alpha.

```
public float a { readonly get; set; }
```

### Property Value

[float](#)

### color

```
public Color color { get; }
```

### Property Value

Color

**h**

Hue.

```
public float h { readonly get; set; }
```

Property Value

[float](#)

**hsv**

```
public HSV hsv { get; }
```

Property Value

[HSV](#)

|

Lightness.

```
public float l { readonly get; set; }
```

Property Value

[float](#)

**s**

Saturation.

```
public float s { readonly get; set; }
```

Property Value

[float](#)

## Methods

### ToColor()

```
public Color ToColor()
```

Returns

Color

### ToHSV()

```
public HSV ToHSV()
```

Returns

[HSV](#)

### ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

[string](#)

The fully qualified type name.

### ToString(int)

```
public string ToString(int decimalPlaces)
```

Parameters

decimalPlaces [int](#)

Returns

[string](#)

## Operators

### explicit operator Color(HSL)

```
public static explicit operator Color(HSL hsl)
```

Parameters

hsl [HSL](#)

Returns

Color

### explicit operator HSV(HSL)

```
public static explicit operator HSV(HSL hsl)
```

Parameters

hsl [HSL](#)

Returns

[HSV](#)

# Struct HSV

Namespace: [PAC.Colour](#)

Represents a colour in HSV form.

```
public struct HSV
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [object.Equals\(object, object\)](#) ,  
[object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Constructors

### HSV(Color)

```
public HSV(Color rgb)
```

Parameters

rgb Color

### HSV(float, float, float)

```
public HSV(float hue, float saturation, float value)
```

Parameters

hue [float](#)

saturation [float](#)

`value` [float](#)

## HSV(float, float, float, float)

```
public HSV(float hue, float saturation, float value, float alpha)
```

### Parameters

`hue` [float](#)

`saturation` [float](#)

`value` [float](#)

`alpha` [float](#)

## Properties

### a

Alpha.

```
public float a { readonly get; set; }
```

### Property Value

[float](#)

### color

```
public Color color { get; }
```

### Property Value

Color

**h**

Hue.

```
public float h { readonly get; set; }
```

Property Value

[float](#)

**hsl**

```
public HSL hsl { get; }
```

Property Value

[HSL](#)

**s**

Saturation.

```
public float s { readonly get; set; }
```

Property Value

[float](#)

**v**

Value.

```
public float v { readonly get; set; }
```

Property Value

[float](#)

## Methods

### ToColor()

```
public Color ToColor()
```

Returns

Color

### ToHSL()

```
public HSL ToHSL()
```

Returns

[HSL](#)

### ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

[string](#)

The fully qualified type name.

### ToString(int)

```
public string ToString(int decimalPlaces)
```

Parameters

decimalPlaces [int](#)

Returns

[string](#)

## Operators

### explicit operator Color(HSV)

```
public static explicit operator Color(HSV hsv)
```

Parameters

hsv [HSV](#)

Returns

Color

### explicit operator HSL(HSV)

```
public static explicit operator HSL(HSV hsv)
```

Parameters

hsv [HSV](#)

Returns

[HSL](#)

# Namespace PAC.ColourPicker

## Classes

### [ColourPreview](#)

A class to represent colour previews - the boxes on colour pickers that show what colours you have selected.

### [GlobalColourPicker](#)

A class for the main colour picker in the program - the one that appears in the main view.

### [HSLColourPicker](#)

A class for the HSL colour picker.

### [HSLHueSaturationBox](#)

A class for the hue/saturation box of the HSL colour picker.

### [HSLLightnessSlider](#)

A class for the lightness slider of the HSL colour picker.

# Class ColourPreview

Namespace: [PAC.ColourPicker](#)

A class to represent colour previews - the boxes on colour pickers that show what colours you have selected.

```
public class ColourPreview : MonoBehaviour
```

## Inheritance

[object](#) ← ColourPreview

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### colour

The colour displayed in the colour preview.

```
public Color colour { get; }
```

#### Property Value

Color

### outlineThickness

```
public float outlineThickness { get; }
```

#### Property Value

[float](#)

# Methods

## SetColour(Color)

```
public void SetColour(Color colour)
```

### Parameters

`colour` Color

## SubscribeToOnDeselect(UnityAction)

Event invoked when colour preview is deselected.

```
public void SubscribeToOnDeselect(UnityAction call)
```

### Parameters

`call` UnityAction

## SubscribeToOnSelect(UnityAction)

Event invoked when colour preview is selected.

```
public void SubscribeToOnSelect(UnityAction call)
```

### Parameters

`call` UnityAction

## SubscribeToOnToggle(UnityAction)

Event invoked when colour preview is selected or deselected.

```
public void SubscribeToOnToggle(UnityAction call)
```

## Parameters

**call** UnityAction

# Class GlobalColourPicker

Namespace: [PAC.ColourPicker](#)

A class for the main colour picker in the program - the one that appears in the main view.

```
public class GlobalColourPicker : MonoBehaviour
```

## Inheritance

[object](#) ← GlobalColourPicker

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### colour

The currently-selected colour.

```
public Color colour { get; }
```

### Property Value

Color

### numOfColourPreviews

```
public int numOfColourPreviews { get; }
```

### Property Value

[int](#)

## primaryColour

The current primary colour.

```
public Color primaryColour { get; }
```

Property Value

Color

## secondaryColour

The current secondary colour.

```
public Color secondaryColour { get; }
```

Property Value

Color

## Methods

### GetColour(int)

Gets the chosen colour at the given index: 0 - primary; 1 - secondary.

```
public Color GetColour(int colourPreviewIndex)
```

Parameters

colourPreviewIndex [int ↗](#)

Returns

Color

## SetColour(Color)

Sets the current colour.

```
public void SetColour(Color colour)
```

Parameters

`colour` Color

## SubscribeToOnColourChange(UnityAction)

Event is invoked when the selected colour changes.

```
public void SubscribeToOnColourChange(UnityAction call)
```

Parameters

`call` UnityAction

# Class HSLColourPicker

Namespace: [PAC.ColourPicker](#)

A class for the HSL colour picker.

```
public class HSLColourPicker : MonoBehaviour
```

## Inheritance

[object](#) ← HSLColourPicker

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### alpha

```
public float alpha { get; }
```

Property Value

[float](#)

### color

```
public Color color { get; }
```

Property Value

Color

### hsl

```
public HSL hsl { get; }
```

Property Value

[HSL](#)

hue

```
public float hue { get; }
```

Property Value

[float](#)

lightness

```
public float lightness { get; }
```

Property Value

[float](#)

mouseSensitivity

```
public float mouseSensitivity { get; }
```

Property Value

[float](#)

saturation

```
public float saturation { get; }
```

Property Value

[float](#)

## slowSensitivityScalar

The sensitivity of the mouse when holding the modifier keyboard shortcut to reduce the sensitivity.

```
public float slowSensitivityScalar { get; }
```

Property Value

[float](#)

## Methods

### SetColour(Color)

```
public void SetColour(Color colour)
```

Parameters

`colour` Color

### SubscribeToOnColourChange(UnityAction)

Event invoked when the selected colour changes.

```
public void SubscribeToOnColourChange(UnityAction call)
```

Parameters

call UnityAction

## UpdateColour()

```
public void UpdateColour()
```

# Class HSLHueSaturationBox

Namespace: [PAC.ColourPicker](#)

A class for the hue/saturation box of the HSL colour picker.

```
public class HSLHueSaturationBox : MonoBehaviour
```

## Inheritance

[object](#) ↗ HSLHueSaturationBox

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### hue

```
public float hue { get; }
```

Property Value

[float](#) ↗

### saturation

```
public float saturation { get; }
```

Property Value

[float](#) ↗

## Methods

## SetHue(float)

```
public void SetHue(float hue)
```

### Parameters

hue [float](#)

## SetSaturation(float)

```
public void SetSaturation(float saturation)
```

### Parameters

saturation [float](#)

# Class HSLLightnessSlider

Namespace: [PAC.ColourPicker](#)

A class for the lightness slider of the HSL colour picker.

```
public class HSLLightnessSlider : MonoBehaviour
```

## Inheritance

[object](#) ← HSLLightnessSlider

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### lightness

```
public float lightness { get; }
```

Property Value

[float](#)

## Methods

### SetDisplayHueSaturation(float, float)

Sets the hue/saturation to use for displaying the lightness gradient.

```
public void SetDisplayHueSaturation(float hue, float saturation)
```

## Parameters

hue [float](#)

**saturation** [float](#)

## SetLightness(float)

```
public void SetLightness(float lightness)
```

### Parameters

**lightness** [float](#)

# Namespace PAC.Config

## Classes

[Colours](#)

[Files](#)

[Tools](#)

# Class Colours

Namespace: [PAC.Config](#)

```
public static class Colours
```

## Inheritance

[object](#) ← Colours

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Fields

### brushOutlineDark

```
public static readonly Color brushOutlineDark
```

Field Value

Color

### brushOutlineLight

```
public static readonly Color brushOutlineLight
```

Field Value

Color

### mask

```
public static readonly Color mask
```

Field Value

Color

**transparent**

```
public static readonly Color32 transparent
```

Field Value

Color32

# Class Files

Namespace: [PAC.Config](#)

```
public static class Files
```

## Inheritance

[object](#) ← Files

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Fields

### fileFormatVersion

```
public static readonly SemanticVersion fileFormatVersion
```

#### Field Value

[SemanticVersion](#)

### pacFileExtension

```
public const string pacFileExtension = "pac"
```

#### Field Value

[string](#)

# Class Tools

Namespace: [PAC.Config](#)

```
public static class Tools
```

## Inheritance

[object](#) ← Tools

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Fields

### maxBrushSize

```
public const int maxBrushSize = 25
```

Field Value

[int](#)

### minBrushSize

```
public const int minBrushSize = 1
```

Field Value

[int](#)

# Namespace PAC.DataStructures

## Classes

### [CustomStack<T>](#)

A custom implementation of a stack to allow removal of items at a specific index.

### [SemanticVersion.JsonConverter](#)

## Structs

### [IntRange](#)

A set of consecutive integers.

### [IntRect](#)

A non-empty rectangular region of integer coordinates.

### [IntVector2](#)

A 2-dimensional vector with integer coordinates.

### [Rational](#)

A fraction  $a / b$  (where  $a$  and  $b$  are integers) kept in simplest form.

### [SemanticVersion](#)

# Class CustomStack<T>

Namespace: [PAC.DataStructures](#)

A custom implementation of a stack to allow removal of items at a specific index.

```
public class CustomStack<T>
```

## Type Parameters

T

### Inheritance

[object](#) ← CustomStack<T>

### Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

### Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### Count

```
public int Count { get; }
```

### Property Value

[int](#)

## Methods

## Clear()

Removes all items from the stack.

```
public void Clear()
```

## Peek()

Returns the item on top of the stack.

```
public T Peek()
```

Returns

T

## Pop()

Removes and returns the item on top of the stack.

```
public T Pop()
```

Returns

T

## Push(T)

Adds the item to the top of the stack.

```
public void Push(T item)
```

Parameters

item T

## Remove(T)

Removes the first occurrence (starting from the top) of the item in the stack.

```
public bool Remove(T item)
```

Parameters

`item` T

Returns

`bool`

true if the item is successfully removed.

## RemoveAll(T)

Removes all occurrences of the item in the stack.

```
public void RemoveAll(T item)
```

Parameters

`item` T

## RemoveAt(int)

Removes the item at the given index and returns it.

```
public T RemoveAt(int index)
```

Parameters

`index` int

Returns

T

## ToArray()

```
public T[] ToArray()
```

Returns

T[]

## ToList()

```
public List<T> ToList()
```

Returns

List ↗<T>

# Struct IntRange

Namespace: [PAC.DataStructures](#)

A set of consecutive integers.

```
public readonly struct IntRange : IReadOnlyList<int>, IReadOnlyContains<int>,
IReadOnlyCollection<int>, IEnumerable<int>, IEnumerable, IEquatable<IntRange>,
ISetComparable<IntRange>
```

## Implements

[IReadOnlyList<int>](#), [IReadOnlyContains<int>](#), [IReadOnlyCollection<int>](#),  
[IEnumerable<int>](#), [IEnumerable](#), [IEquatable<IntRange>](#),  
[ISetComparable<IntRange>](#)

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>,  
IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>,  
IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>,  
Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>,  
Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>,  
params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,

[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Product\(IEnumerable<int>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#) ,  
 [IReadOnlyListExtensions.GetRange<T>\( IReadOnlyList<T>, int, int\)](#) ,  
 [IReadOnlyListExtensions.GetRange<T>\( IReadOnlyList<T>, Range\)](#) ,  
 [IntRangeExtensions.GetRange<T>\( IReadOnlyList<T>, IntRange\)](#)

## Remarks

The enumerator iterates through the elements from [startElement](#) to [endElement](#).

If `startBoundary` and `endBoundary` are equal but their being inclusive / exclusive do not match, we define the range to be empty. This definition is useful in situations such as

```
foreach (int index in IntRange.InclExcl(0, array.Length))
{
    Console.WriteLine($"Element {index}: {array[index]}");
}
```

so that it's equivalent to

```
for (int index = 0; index < array.Length; index++)
{
    Console.WriteLine($"Element {index}: {array[index]}");
}
```

## Constructors

### IntRange(int, int, bool, bool)

Creates a new range.

```
public IntRange(int startBoundary, int endBoundary, bool startBoundaryInclusive,
bool endBoundaryInclusive)
```

#### Parameters

`startBoundary` `int`

The start of the range. Can be inclusive or exclusive.

`endBoundary` `int`

The end of the range. Can be inclusive or exclusive.

`startBoundaryInclusive` `bool`

Whether the start boundary is inclusive or exclusive.

`endBoundaryInclusive` `bool`

Whether the end boundary is inclusive or exclusive.

## Remarks

If `startBoundary` and `endBoundary` are equal but their being inclusive / exclusive do not match, we define the range to be empty. See [IntRange](#) for more details.

## Fields

### All

The range from [MinValue](#) (inclusive) to [MaxValue](#) (inclusive).

```
public static readonly IntRange All
```

### Field Value

[IntRange](#)

### Empty

The range from 0 (exclusive) to 0 (exclusive), which contains no elements.

```
public static readonly IntRange Empty
```

### Field Value

[IntRange](#)

### endBoundary

The end of the range. Can be inclusive or exclusive.

```
public readonly int endBoundary
```

### Field Value

[int](#)

## Remarks

Will be different from [endElement](#) if this boundary is exclusive.

## See Also

[endBoundaryInclusive](#), [startBoundary](#)

## endBoundaryInclusive

Whether [endBoundary](#) is inclusive or exclusive.

```
public readonly bool endBoundaryInclusive
```

## Field Value

[bool](#)

## See Also

[startBoundaryInclusive](#)

## startBoundary

The start of the range. Can be inclusive or exclusive.

```
public readonly int startBoundary
```

## Field Value

[int](#)

## Remarks

Will be different from [startElement](#) if this boundary is exclusive.

## See Also

[startBoundaryInclusive](#), [endBoundary](#)

## startBoundaryInclusive

Whether [startBoundary](#) is inclusive or exclusive.

```
public readonly bool startBoundaryInclusive
```

Field Value

[bool](#)

### See Also

[endBoundaryInclusive](#)

## Properties

### Count

The number of elements in the range.

```
public int Count { get; }
```

Property Value

[int](#)

### Remarks

May throw an [OverflowException](#). For example, the [Count](#) of a range from [MinValue](#) to [MaxValue](#) cannot be expressed as an [int](#). Using [LongCount](#) instead will never encounter this issue.

### Exceptions

[OverflowException](#)

The number of elements is too large to represent as an [int](#).

### this[int]

Indexes the elements of the range from [startElement](#) to [endElement](#).

```
public int this[int index] { get; }
```

## Parameters

index [int](#)

## Property Value

[int](#)

## Remarks

Some ranges, such as a range from [MinValue](#) to [MaxValue](#), are too long for every element to be indexed by an [int](#). Using [this\[long\]](#) instead will never have this issue.

## Exceptions

[ArgumentOutOfRangeException](#)

[index](#) is not a valid index in the range.

## this[long]

Indexes the elements of the range from [startElement](#) to [endElement](#).

```
public int this[long index] { get; }
```

## Parameters

index [long](#)

## Property Value

[int](#)

## Remarks

Unlike [this\[int\]](#), every element in a range can be indexed with a [long](#). For example, the range from [MinValue](#) to [MaxValue](#) is too long for every element to be indexed by an [int](#), but every element can be indexed by a [long](#).

# Exceptions

## [ArgumentOutOfRangeException](#)

`index` is not a valid index in the range.

# LongCount

The same as [Count](#) but computed as a [long](#).

```
public long LongCount { get; }
```

Property Value

[long](#)

# Remarks

Unlike [Count](#), this will never throw an [OverflowException](#). For example, the largest range, [MinValue](#) to [MaxValue](#), cannot be expressed as an [int](#) but can be expressed as a [long](#).

# asDecreasing

A range with the same elements, in decreasing order.

```
public IntRange asDecreasing { get; }
```

Property Value

[IntRange](#)

Either the range unchanged or [reverse](#).

# See Also

[asIncreasing](#)

# asExclExcl

A range with the same elements, in the same order, but expressed with [startBoundary](#) exclusive and [endBoundary](#) exclusive.

```
public IntRange asExclExcl { get; }
```

Property Value

[IntRange](#)

**See Also**

[asExclIncl](#), [asInclExcl](#), [asInclIncl](#)

## asExclIncl

A range with the same elements, in the same order, but expressed with [startBoundary](#) exclusive and [endBoundary](#) inclusive.

```
public IntRange asExclIncl { get; }
```

Property Value

[IntRange](#)

**See Also**

[asExclExcl](#), [asInclExcl](#), [asInclIncl](#)

## asInclExcl

A range with the same elements, in the same order, but expressed with [startBoundary](#) inclusive and [endBoundary](#) exclusive.

```
public IntRange asInclExcl { get; }
```

Property Value

[IntRange](#)

**See Also**

[asExclExcl](#), [asExclIncl](#), [asInclIncl](#)

## asInclIncl

A range with the same elements, in the same order, but expressed with [startBoundary](#) inclusive and [endBoundary](#) inclusive.

```
public IntRange asInclIncl { get; }
```

Property Value

[IntRange](#)

Exceptions

[InvalidOperationException](#)

The range is empty.

### See Also

[asExclExcl](#), [asExclIncl](#), [asInclExcl](#)

## asIncreasing

A range with the same elements, in increasing order.

```
public IntRange asIncreasing { get; }
```

Property Value

[IntRange](#)

Either the range unchanged or [reverse](#).

### See Also

[asDecreasing](#)

## endElement

The last element when iterating through the range (from start to end).

```
public int endElement { get; }
```

Property Value

[int](#)

Remarks

Will be different from [endBoundary](#) if that boundary is exclusive.

### See Also

[startElement](#)

## isEmpty

Whether the range has no elements.

```
public bool isEmpty { get; }
```

Property Value

[bool](#)

## isExclExcl

Whether [startBoundary](#) is exclusive and [endBoundary](#) is exclusive.

```
public bool isExclExcl { get; }
```

Property Value

[bool](#)

Remarks

Equivalent to `!startBoundaryInclusive && !endBoundaryInclusive`.

## See Also

[isExclIncl](#), [isInclExcl](#), [isInclIncl](#)

## isExclIncl

Whether [startBoundary](#) is exclusive and [endBoundary](#) is inclusive.

```
public bool isExclIncl { get; }
```

Property Value

[bool](#)

## Remarks

Equivalent to `!startBoundaryInclusive && endBoundaryInclusive`.

## See Also

[isExclExcl](#), [isInclExcl](#), [isInclIncl](#)

## isInclExcl

Whether [startBoundary](#) is inclusive and [endBoundary](#) is exclusive.

```
public bool isInclExcl { get; }
```

Property Value

[bool](#)

## Remarks

Equivalent to `startBoundaryInclusive && !endBoundaryInclusive`.

## See Also

[isExclExcl](#), [isExclIncl](#), [isInclIncl](#)

## isInclIncl

Whether [startBoundary](#) is inclusive and [endBoundary](#) is inclusive.

```
public bool isInclIncl { get; }
```

Property Value

[bool](#)

Remarks

Equivalent to [startBoundaryInclusive](#) && [endBoundaryInclusive](#).

**See Also**

[isExclExcl](#), [isExclIncl](#), [isInclExcl](#)

## maxBoundary

The higher of [startBoundary](#) and [endBoundary](#).

```
public int maxBoundary { get; }
```

Property Value

[int](#)

Remarks

Will be different from [maxElement](#) if this boundary is exclusive.

**See Also**

[maxBoundaryInclusive](#), [minBoundary](#)

## maxBoundaryInclusive

Whether [maxBoundary](#) is inclusive or exclusive.

```
public bool maxBoundaryInclusive { get; }
```

Property Value

[bool](#)

Remarks

If [startBoundary](#) and [endBoundary](#) are equal but their being inclusive / exclusive do not match, we define this to be exclusive. See [IntRange](#) for more details.

**See Also**

[minBoundaryInclusive](#)

## maxElement

The highest element in the range.

```
public int maxElement { get; }
```

Property Value

[int](#)

Remarks

Will be different from [maxBoundary](#) if that boundary is exclusive.

**See Also**

[minElement](#)

## minBoundary

The lower of [startBoundary](#) and [endBoundary](#).

```
public int minBoundary { get; }
```

Property Value

[int](#)

## Remarks

Will be different from [minElement](#) if this boundary is exclusive.

## See Also

[minBoundaryInclusive](#), [maxBoundary](#)

## minBoundaryInclusive

Whether [minBoundary](#) is inclusive or exclusive.

```
public bool minBoundaryInclusive { get; }
```

## Property Value

[bool](#)

## Remarks

If [startBoundary](#) and [endBoundary](#) are equal but their being inclusive / exclusive do not match, we define this to be exclusive. See [IntRange](#) for more details.

## See Also

[maxBoundaryInclusive](#)

## minElement

The lowest element in the range.

```
public int minElement { get; }
```

## Property Value

[int](#)

## Remarks

Will be different from [minBoundary](#) if that boundary is exclusive.

## See Also

## [maxElement](#)

### reverse

The range with [startBoundary](#) and [endBoundary](#) swapped, and [startBoundaryInclusive](#) and [endBoundaryInclusive](#) swapped.

```
public IntRange reverse { get; }
```

Property Value

[IntRange](#)

### startElement

The first element when iterating through the range (from start to end).

```
public int startElement { get; }
```

Property Value

[int](#)

### Remarks

Will be different from [startBoundary](#) if that boundary is exclusive.

### See Also

[endElement](#)

## Methods

### AtLeast(int)

Creates a range from [x](#) (inclusive) to [MaxValue](#) (inclusive).

```
public static IntRange AtLeast(int x)
```

Parameters

x [int](#)

Returns

[IntRange](#)

**See Also**

[AtMost\(int\)](#), [LessThan\(int\)](#), [MoreThan\(int\)](#)

## AtMost(int)

Creates a range from x (inclusive) to [MinValue](#) (inclusive).

```
public static IntRange AtMost(int x)
```

Parameters

x [int](#)

Returns

[IntRange](#)

**See Also**

[LessThan\(int\)](#), [AtLeast\(int\)](#), [MoreThan\(int\)](#)

## BoundingRange(IEnumerable<int>)

Returns the smallest range containing all the given values.

```
public static IntRange BoundingRange(IEnumerable<int> integers)
```

Parameters

integers [IEnumerable](#)<[int](#)>

Returns

## [IntRange](#)

The smallest range containing all the given values. It will be expressed with both boundaries inclusive, unless the sequence of values is empty, in which case it will be expressed with both boundaries exclusive.

Exceptions

## [ArgumentNullException](#)

`integers` is null.

## Clamp(int)

Clamps the given integer to be within the [IntRange](#).

```
public int Clamp(int n)
```

Parameters

### `n` [int](#)

The value to clamp.

Returns

### [int](#)

- `n` if [minElement](#) <= `n` <= [maxElement](#)
- [minElement](#) if `n` < [minElement](#)
- [maxElement](#) if [maxElement](#) < `n`

Exceptions

## [InvalidOperationException](#)

The range is empty.

## See Also

### [Clamp](#)([int](#), [int](#), [int](#))

## Contains(int)

Whether the given integer is in the range.

```
public bool Contains(int x)
```

Parameters

x [int](#)

Returns

[bool](#)

## Equals(IntRange)

The same as [operator ==\(IntRange, IntRange\)](#).

```
public bool Equals(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

### See Also

[SequenceEquals\(IntRange\)](#), [SetEquals\(IntRange\)](#)

## Equals(object)

The same as [Equals\(IntRange\)](#).

```
public override bool Equals(object obj)
```

Parameters

`obj object`

Returns

`bool`

**See Also**

[SequenceEquals\(IntRange\)](#), [SetEquals\(IntRange\)](#).

## ExclExcl(int, int)

Creates a range from `startBoundary` (exclusive) to `endBoundary` (exclusive).

```
public static IntRange ExclExcl(int startBoundary, int endBoundary)
```

Parameters

`startBoundary int`

`endBoundary int`

Returns

[IntRange](#)

**See Also**

[ExclIncl\(int, int\)](#), [InclExcl\(int, int\)](#), [InclIncl\(int, int\)](#)

## ExclIncl(int, int)

Creates a range from `startBoundary` (exclusive) to `endBoundary` (inclusive).

```
public static IntRange ExclIncl(int startBoundary, int endBoundary)
```

Parameters

`startBoundary int`

`endBoundary` [int](#)

Returns

[IntRange](#)

Remarks

If `startBoundary` and `endBoundary` are equal, we define the range to be empty. See [IntRange](#) for more details.

### See Also

[ExclExcl\(int, int\)](#), [InclExcl\(int, int\)](#), [InclIncl\(int, int\)](#)

## Extend(int, int)

Adds `startBoundaryOffset` to [startBoundary](#) and adds `endBoundaryOffset` to [endBoundary](#).

```
public IntRange Extend(int startBoundaryOffset, int endBoundaryOffset)
```

Parameters

`startBoundaryOffset` [int](#)

`endBoundaryOffset` [int](#)

Returns

[IntRange](#)

Remarks

Preserves [startBoundaryInclusive](#) and [endBoundaryInclusive](#).

## GetEnumerator()

Iterates through the elements from [startElement](#) to [endElement](#).

```
public IEnumerator<int> GetEnumerator()
```

Returns

[IEnumerator](#) <int>

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## InclExcl(int, int)

Creates a range from `startBoundary` (inclusive) to `endBoundary` (exclusive).

```
public static IntRange InclExcl(int startBoundary, int endBoundary)
```

Parameters

`startBoundary` [int](#)

`endBoundary` [int](#)

Returns

[IntRange](#)

Remarks

If `startBoundary` and `endBoundary` are equal, we define the range to be empty. See [IntRange](#) for more details.

### See Also

[ExclExcl\(int, int\)](#), [ExclIncl\(int, int\)](#), [InclIncl\(int, int\)](#)

## InclIncl(int, int)

Creates a range from `startBoundary` (inclusive) to `endBoundary` (inclusive).

```
public static IntRange InclIncl(int startBoundary, int endBoundary)
```

Parameters

`startBoundary` [int](#)

`endBoundary` [int](#)

Returns

[IntRange](#)

### See Also

[ExclExcl\(int, int\)](#), [ExclIncl\(int, int\)](#), [InclExcl\(int, int\)](#)

## Intersect(IntRange, IntRange)

Whether the two ranges have any elements in common.

```
public static bool Intersect(IntRange a, IntRange b)
```

Parameters

`a` [IntRange](#)

`b` [IntRange](#)

Returns

[bool](#)

### See Also

[Intersects\(IntRange\)](#)

## Intersection(IntRange)

Returns the elements the two ranges have in common.

```
public IntRange Intersection(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[IntRange](#)

The intersection of the two ranges, expressed with both boundaries inclusive, unless the intersection is empty, in which case it will be expressed with both boundaries exclusive.

**See Also**

[Intersection\(IntRange, IntRange\)](#)

## Intersection(IntRange, IntRange)

Returns the elements the two ranges have in common.

```
public static IntRange Intersection(IntRange a, IntRange b)
```

Parameters

a [IntRange](#)

b [IntRange](#)

Returns

[IntRange](#)

The intersection of the two ranges. It will be expressed with both boundaries inclusive, unless the intersection is empty, in which case it will be expressed with both boundaries exclusive.

**See Also**

[Intersection\(IntRange\)](#)

## Intersects(IntRange)

Whether the two ranges have any elements in common.

```
public bool Intersects(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

### See Also

[Intersect\(IntRange, IntRange\)](#)

## IsProperSubsequenceOf(IntRange)

```
public bool IsProperSubsequenceOf(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

## IsProperSupersequenceOf(IntRange)

```
public bool IsProperSupersequenceOf(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

## IsSubsequenceOf(IntRange)

`public bool IsSubsequenceOf(IntRange other)`

Returns

[Parameters](#)

[other](#) [IntRange](#)

Parameters

[bool](#)

## IsSubsetOf(IntRange)

Whether [this](#) as a set is a subset of [other](#) as a set.

`public bool IsSubsetOf(IntRange other)`

Returns

[Parameters](#)

[other](#) [IntRange](#)

Parameters

[bool](#)

## Remarks

Recall that '[this](#) is subset of [other'](#) means that every element of [this](#) is an element of [other](#), ignoring order and duplicate elements.

If [T1](#) implements [ISetComparable<T2>](#) and [T2](#) implements [ISetComparable<T1>](#), then [T1.IsSubsetOf\(T2\)](#) should be logically equivalent to [T2.IsSupersetOf\(T1\)](#).

## See Also

## [IsSupersetOf\(T\)](#)

### IsSupersequenceOf(IntRange)

```
public bool IsSupersequenceOf(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

### IsSupersetOf(IntRange)

Whether [this](#) as a set is a superset of other as a set.

```
public bool IsSupersetOf(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

Remarks

Recall that '[this](#) is superset of other' means that every element of other is an element of [this](#), ignoring order and duplicate elements.

If T1 implements [ISetComparable<T2>](#) and T2 implements [ISetComparable<T1>](#), then [T1.IsSupersetOf\(T2\)](#) should be logically equivalent to [T2.IsSubsetOf\(T1\)](#).

## See Also

[IsSubsetOf\(T\)](#)

## LessThan(int)

Creates a range from `x` (exclusive) to [MinValue](#) (inclusive).

```
public static IntRange LessThan(int x)
```

Parameters

`x` [int](#)

Returns

[IntRange](#)

### See Also

[AtMost\(int\)](#), [AtLeast\(int\)](#), [MoreThan\(int\)](#).

## MoreThan(int)

Creates a range from `x` (exclusive) to [MaxValue](#) (inclusive).

```
public static IntRange MoreThan(int x)
```

Parameters

`x` [int](#)

Returns

[IntRange](#)

### See Also

[AtMost\(int\)](#), [LessThan\(int\)](#), [AtLeast\(int\)](#).

## SequenceEqual(IntRange, IntRange)

Whether the two ranges have exactly the same elements in the same order.

```
public static bool SequenceEqual(IntRange a, IntRange b)
```

Parameters

a [IntRange](#)

b [IntRange](#)

Returns

[bool](#)

Remarks

Note that ranges with different boundaries can give the same sequence (and therefore be considered sequence-equal), due to the boundaries being inclusive or exclusive.

**See Also**

[Equals\(IntRange\)](#), [SetEquals\(IntRange\)](#)

## SequenceEquals(IntRange)

Whether the two ranges have exactly the same elements in the same order.

```
public bool SequenceEquals(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

Remarks

Note that ranges with different boundaries can give the same sequence (and therefore be considered sequence-equal), due to the boundaries being inclusive or exclusive.

**See Also**

[Equals\(IntRange\)](#), [SetEquals\(IntRange\)](#)

## SetEqual(IntRange, IntRange)

Whether the two ranges have exactly the same elements, ignoring order.

```
public static bool SetEqual(IntRange a, IntRange b)
```

Parameters

a [IntRange](#)

b [IntRange](#)

Returns

[bool](#)

Remarks

Note that ranges with different boundaries can have the same elements (and therefore be considered set-equal), due to the boundaries being inclusive or exclusive.

### See Also

[Equals\(IntRange\)](#), [SequenceEquals\(IntRange\)](#)

## SetEquals(IntRange)

Whether the two ranges have exactly the same elements, ignoring order.

```
public bool SetEquals(IntRange other)
```

Parameters

other [IntRange](#)

Returns

[bool](#)

## Remarks

Note that ranges with different boundaries can have the same elements (and therefore be considered set-equal), due to the boundaries being inclusive or exclusive.

## See Also

[Equals\(IntRange\)](#), [SequenceEquals\(IntRange\)](#).

## Singleton(int)

Creates a range from `x` (inclusive) to `x` (inclusive).

```
public static IntRange Singleton(int x)
```

## Parameters

`x` [int](#)

## Returns

[IntRange](#)

## ToString()

Represents the range as a string using the mathematical notation for intervals - i.e. square brackets denote an inclusive boundary and round brackets denote an exclusive boundary.

For example, the range from -1 (inclusive) to 5 (exclusive) will be represented as "`[-1, 5)`".

```
public override string ToString()
```

## Returns

[string](#)

# Operators

## operator +(IntRange, int)

Translates the range by the given integer.

```
public static IntRange operator +(IntRange range, int integer)
```

Parameters

range [IntRange](#)

integer [int](#)

Returns

[IntRange](#)

Exceptions

[OverflowException](#)

If any translated elements cannot be expressed as an [int](#).

## operator +(int, IntRange)

Translates the range by the given integer.

```
public static IntRange operator +(int integer, IntRange range)
```

Parameters

integer [int](#)

range [IntRange](#)

Returns

[IntRange](#)

# Exceptions

## [OverflowException](#)

If any translated elements cannot be expressed as an [int](#).

## operator ==(IntRange, IntRange)

Whether the two ranges have the same start boundary and end boundary and the same inclusivity at each boundary - i.e. compares the [struct](#)'s' fields.

```
public static bool operator ==(IntRange a, IntRange b)
```

### Parameters

a [IntRange](#)

b [IntRange](#)

### Returns

[bool](#)

### Remarks

Note that having the same elements does not necessarily mean they are ==, due to the boundaries being inclusive or exclusive.

### See Also

[Equals\(IntRange\)](#), [SequenceEquals\(IntRange\)](#), [SetEquals\(IntRange\)](#)

## operator !=(IntRange, IntRange)

Whether the two ranges have the same start boundary and end boundary and the same inclusivity at each boundary - i.e. compares the [struct](#)'s' fields.

```
public static bool operator !=(IntRange a, IntRange b)
```

### Parameters

a [IntRange](#)

b [IntRange](#)

Returns

[bool](#)

Remarks

Note that being `!=` does not necessarily mean they have different elements, due to the boundaries being inclusive or exclusive.

### See Also

[Equals\(IntRange\)](#), [SequenceEquals\(IntRange\)](#), [SetEquals\(IntRange\)](#)

## operator -(IntRange, int)

Translates the range by the given integer.

```
public static IntRange operator -(IntRange range, int integer)
```

Parameters

range [IntRange](#)

integer [int](#)

Returns

[IntRange](#)

Exceptions

[OverflowException](#)

If any translated elements cannot be expressed as an [int](#).

## operator -(int, IntRange)

Subtracts each element of the range from the given integer.

```
public static IntRange operator -(int integer, IntRange range)
```

Parameters

integer [int](#)

range [IntRange](#)

Returns

[IntRange](#)

Exceptions

[OverflowException](#)

If any translated elements cannot be expressed as an [int](#).

## operator -(IntRange)

Negates each element of the range.

```
public static IntRange operator -(IntRange range)
```

Parameters

range [IntRange](#)

Returns

[IntRange](#)

Exceptions

[OverflowException](#)

If any negated elements cannot be expressed as an [int](#). This only occurs if the range contains [MinValue](#).

# Struct IntRect

Namespace: [PAC.DataStructures](#)

A non-empty rectangular region of integer coordinates.

```
public readonly struct IntRect : IReadOnlyContains<IntVector2>,
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>, IEnumerable,
IEquatable<IntRect>, ISetComparable<IntRect>
```

## Implements

[IReadOnlyContains<IntVector2>](#), [IReadOnlyCollection<IntVector2>](#),  
[IEnumerable<IntVector2>](#), [IEnumerable](#), [IEquatable<IntRect>](#),  
[ISetComparable<IntRect>](#)

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,

[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

When we say an [IntVector2](#) is in the rect, we mean that whole grid square is in the rect. For example, the rect with corners (0, 0), (0, 4), (3, 0), (3, 4) is a 4x5 rectangle:

```
(0, 4) -> # # # # <- (3, 4)
# # # #
# # # #
```

```
# # # #
(0, 0) -> # # # # <- (3, 0)
```

The enumerator iterates over the points in the rect, starting with the bottom row, read left to right, then the next row, etc. This ordering is useful since it matches the expected order of pixels in Unity's `Texture2D.SetPixels(Color[])`.

## Constructors

### IntRect(IntRange, IntRange)

Creates a rect with the given ranges of x and y coordinates.

```
public IntRect(IntRange xRange, IntRange yRange)
```

#### Parameters

`xRange` [IntRange](#)

The range of x coordinates.

`yRange` [IntRange](#)

The range of y coordinates.

#### Remarks

The x and y ranges must be non-empty, as [IntRect](#)s are non-empty.

#### Exceptions

[ArgumentException](#)

`xRange` is empty and/or `yRange` is empty.

### IntRect(IntVector2, IntVector2)

Creates an [IntRect](#) with the two given points as opposite corners (inclusive) of the rect.

```
public IntRect(IntVector2 corner, IntVector2 oppositeCorner)
```

## Parameters

`corner` [IntVector2](#)

The corner diagonally opposite `oppositeCorner`.

`oppositeCorner` [IntVector2](#)

The corner diagonally opposite `corner`.

## Remarks

For example, with `corner = (3, 0)` and `oppositeCorner = (0, 4)`, the rect will have corners `(0, 0)`, `(0, 4)`, `(3, 0)`, `(3, 4)`:

```
(0, 4) -> # # # # <- (3, 4)
# # #
# # #
# # #
(0, 0) -> # # # # <- (3, 0)
```

## Fields

`bottomLeft`

The coordinates of the bottom-left point in the rect (inclusive).

```
public readonly IntVector2 bottomLeft
```

## Field Value

[IntVector2](#)

## See Also

[topRight](#), [bottomRight](#), [topLeft](#)

`topRight`

The coordinates of the top-right point in the rect (inclusive).

```
public readonly IntVector2 topRight
```

Field Value

[IntVector2](#)

**See Also**

[bottomLeft](#), [bottomRight](#), [topLeft](#)

## Properties

### Count

The number of points in the rect.

```
public int Count { get; }
```

Property Value

[int](#)

### bottomRight

The coordinates of the bottom-right point in the rect (inclusive).

```
public IntVector2 bottomRight { get; }
```

Property Value

[IntVector2](#)

**See Also**

[bottomLeft](#), [topRight](#), [topLeft](#)

## height

The number of integer points the rect covers vertically.

```
public int height { get; }
```

Property Value

[int](#)

### See Also

[width, size](#)

## isSquare

Whether the rect is a square - i.e. the [width](#) and [height](#) are equal.

```
public bool isSquare { get; }
```

Property Value

[bool](#)

### See Also

[IsSquare\(IntRect\)](#)

## maxX

The maximum x coordinate of points in the rect.

```
public int maxX { get; }
```

Property Value

[int](#)

### See Also

[minX, minY, maxY](#)

## maxY

The maximum y coordinate of points in the rect.

```
public int maxY { get; }
```

Property Value

[int ↗](#)

### See Also

[minX](#), [maxX](#), [minY](#)

## minX

The minimum x coordinate of points in the rect.

```
public int minX { get; }
```

Property Value

[int ↗](#)

### See Also

[maxX](#), [minY](#), [maxY](#)

## minY

The minimum y coordinate of points in the rect.

```
public int minY { get; }
```

Property Value

[int ↗](#)

### See Also

[minX](#), [maxX](#), [maxY](#)

## size

The vector ([width](#), [height](#)).

```
public IntVector2 size { get; }
```

Property Value

[IntVector2](#)

### See Also

[width](#), [height](#)

## topLeft

The coordinates of the top-left point in the rect (inclusive).

```
public IntVector2 topLeft { get; }
```

Property Value

[IntVector2](#)

### See Also

[bottomLeft](#), [topRight](#), [bottomRight](#)

## width

The number of integer points the rect covers horizontally.

```
public int width { get; }
```

Property Value

[int](#)

### See Also

[height](#), [size](#)

## xRange

The inclusive range of x coordinates in the rect, in increasing order.

```
public IntRange xRange { get; }
```

Property Value

[IntRange](#)

### See Also

[yRange](#)

## yRange

The inclusive range of y coordinates in the rect, in increasing order.

```
public IntRange yRange { get; }
```

Property Value

[IntRange](#)

## Methods

### BoundingRect(IntRect, IntRect)

Returns the smallest rect containing both the given rects.

```
public static IntRect BoundingRect(IntRect a, IntRect b)
```

Parameters

a [IntRect](#)

b [IntRect](#)

Returns

## [IntRect](#)

### BoundingRect(params IntRect[])

Returns the smallest rect containing all the given rects.

```
public static IntRect BoundingRect(params IntRect[] rects)
```

Parameters

rects [IntRect\[\]](#)

Returns

[IntRect](#)

Exceptions

[ArgumentNullException](#)

rects is null.

[ArgumentException](#)

rects is empty.

### BoundingRect(params IntVector2[])

Returns the smallest rect containing all the given vectors.

```
public static IntRect BoundingRect(params IntVector2[] vectors)
```

Parameters

vectors [IntVector2\[\]](#)

Returns

[IntRect](#)

## Exceptions

### [ArgumentNullException](#)

`vectors` is null.

### [ArgumentException](#)

`vectors` is empty.

## BoundingRect(IEnumerable<IntRect>)

Returns the smallest rect containing all the given rects.

```
public static IntRect BoundingRect(IEnumerable<IntRect> rects)
```

## Parameters

### `rects` [IEnumerable](#)<IntRect>

## Returns

### [IntRect](#)

## Exceptions

### [ArgumentNullException](#)

`rects` is null.

### [ArgumentException](#)

`rects` is empty.

## BoundingRect(IEnumerable<IntVector2>)

Returns the smallest rect containing all the given vectors.

```
public static IntRect BoundingRect(IEnumerable<IntVector2> vectors)
```

Parameters

`vectors` [IEnumerable](#)<[IntVector2](#)>

Returns

[IntRect](#)

Exceptions

[ArgumentNullException](#)

`vectors` is null.

[ArgumentException](#)

`vectors` is empty.

## Clamp(IntVector2)

Returns the point in the rect that is closest to the given vector. Equivalently, it clamps the vector's x coord to be in the rect's range of x coords, and clamps the vector's y coord to be in the rect's range of y coords.

```
public IntVector2 Clamp(IntVector2 vector)
```

Parameters

`vector` [IntVector2](#)

Returns

[IntVector2](#)

### See Also

[Clamp](#)([int](#), [int](#), [int](#))

## Contains(IntVector2)

Whether the point is in the rect.

```
public bool Contains(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#)

**See Also**

[Contains\(int, int\)](#)

## Contains(int, int)

Whether the point  $(x, y)$  is in the rect.

```
public bool Contains(int x, int y)
```

Parameters

x [int](#)

y [int](#)

Returns

[bool](#)

**See Also**

[Contains\(IntVector2\)](#)

## ContainsX(int)

Returns whether the rect contains any points with the given x coord.

```
public bool ContainsX(int x)
```

Parameters

x [int](#)

Returns

[bool](#)

**See Also**

[ContainsY\(int\)](#)

## ContainsY(int)

Returns whether the rect contains any points with the given y coord.

```
public bool ContainsY(int y)
```

Parameters

y [int](#)

Returns

[bool](#)

**See Also**

[ContainsX\(int\)](#)

## Equals(IntRect)

The same as [operator ==\(IntRect, IntRect\)](#).

```
public bool Equals(IntRect other)
```

Parameters

other [IntRect](#)

Returns

[bool](#)

## Equals(object)

The same as [Equals\(IntRect\)](#).

```
public override bool Equals(object obj)
```

Parameters

[obj](#) [object](#)

Returns

[bool](#)

## FilterPointsInside(IEnumerable<IntVector2>)

Returns the elements of the sequence that are in this rect.

```
public IEnumerable<IntVector2> FilterPointsInside(IEnumerable<IntVector2> vectors)
```

Parameters

[vectors](#) [IEnumerable](#)<[IntVector2](#)>

Returns

[IEnumerable](#)<[IntVector2](#)>

A lazily-generated sequence containing the elements of [vectors](#) that are in this rect.

## Remarks

Preserves the order of the sequence.

## Exceptions

## [ArgumentNullException](#)

`vectors` is null.

### See Also

[FilterPointsOutside\(IEnumerable<IntVector2>\)](#)

## FilterPointsOutside(IEnumerable<IntVector2>)

Returns the elements of the sequence that are outside this rect.

```
public IEnumerable<IntVector2> FilterPointsOutside(IEnumerable<IntVector2> vectors)
```

### Parameters

`vectors` [IEnumerable](#)<IntVector2>

### Returns

[IEnumerable](#)<IntVector2>

A lazily-generated sequence containing the elements of `vectors` that are not in this rect.

### Remarks

Preserves the order of the sequence.

### Exceptions

[ArgumentNullException](#)

`vectors` is null.

### See Also

[FilterPointsInside\(IEnumerable<IntVector2>\)](#)

## Flip(FlipAxis)

Returns the rect flipped across the given axis.

```
public IntRect Flip(FlipAxis axis)
```

Parameters

**axis** [FlipAxis](#)

Returns

[IntRect](#)

**See Also**

[Flip\(FlipAxis\)](#)

## GetEnumerator()

Iterates over the points in the rect, starting with the bottom row, read left to right, then the next row, etc.

```
public IEnumerator<IntVector2> GetEnumerator()
```

Returns

[IEnumerator](#) <[IntVector2](#)>

Remarks

The ordering of this enumerator is useful since it matches the expected order of pixels in Unity's `Texture2D.SetPixels(Color[])`.

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## Intersection(IntRect)

Returns the set intersection of the two rects.

```
public IntRect Intersection(IntRect other)
```

Parameters

other [IntRect](#)

Returns

[IntRect](#)

Exceptions

[InvalidOperationException](#)

This rect and other do not intersect.

### See Also

[Intersection\(IntRect, IntRect\)](#)

## Intersection(IntRect, IntRect)

Returns the set intersection of the two rects.

```
public static IntRect Intersection(IntRect a, IntRect b)
```

Parameters

a [IntRect](#)

b [IntRect](#)

Returns

## [IntRect](#)

### Exceptions

#### [InvalidOperationException](#)

a and b do not intersect.

### See Also

#### [Intersection\(IntRect\)](#)

## IsSquare(IntRect)

Whether the rect is a square - i.e. the [width](#) and [height](#) are equal.

```
public static bool IsSquare(IntRect rect)
```

### Parameters

rect [IntRect](#)

### Returns

[bool](#)

### See Also

#### [isSquare](#)

## IsSubsetOf(IntRect)

Whether this rect is a subset of other.

```
public bool IsSubsetOf(IntRect other)
```

### Parameters

other [IntRect](#)

### Returns

[bool](#)

## See Also

[IsSupersetOf\(IntRect\)](#).

# IsSupersetOf(IntRect)

Whether this rect is a superset of `other`.

```
public bool IsSupersetOf(IntRect other)
```

## Parameters

`other` [IntRect](#)

## Returns

[bool](#)

## See Also

[IsSubsetOf\(IntRect\)](#).

# Overlap(IntRect, IntRect)

Whether the two rects have any points in common.

```
public static bool Overlap(IntRect a, IntRect b)
```

## Parameters

`a` [IntRect](#)

`b` [IntRect](#)

## Returns

[bool](#)

## See Also

## [Overlaps\(IntRect\)](#)

# Overlaps(IntRect)

Whether the two rects have any points in common.

```
public bool Overlaps(IntRect rect)
```

Parameters

rect [IntRect](#)

Returns

[bool](#)

## See Also

[Overlap\(IntRect, IntRect\)](#)

# RandomPoint(Random)

Generates a uniformly random point within the rect.

```
public IntVector2 RandomPoint(Random random)
```

Parameters

random [Random](#)

Returns

[IntVector2](#)

# RandomSubRect(Random)

Returns the rect defined by two independently uniformly randomly generated points within the rect.

```
public IntRect RandomSubRect(Random random)
```

Parameters

random [Random](#) ↗

Returns

[IntRect](#)

## Rotate(RotationAngle)

Returns the rect rotated clockwise by the given angle.

```
public IntRect Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

[IntRect](#)

### See Also

[Rotate\(RotationAngle\)](#)

## SetEquals(IntRect)

Whether the two rects describe the same set of points.

```
public bool SetEquals(IntRect other)
```

Parameters

other [IntRect](#)

Returns

[bool](#)

Remarks

The is the same as [operator ==\(IntRect, IntRect\)](#).

## ToString()

Represents the rect as a string in the form "`((bottom-left x coord, bottom-left y coord), (top-right x coord, top-right y coord))`".

```
public override string ToString()
```

Returns

[string](#)

## TranslateClamp(IntRect)

Translates `toClamp` so it is a subset of this rect. Chooses the smallest such translation.

```
public IntRect TranslateClamp(IntRect toClamp)
```

Parameters

`toClamp` [IntRect](#)

Returns

[IntRect](#)

Remarks

Note this is only possible when `toClamp` is at most as wide and at most as tall as this rect.

Exceptions

## [ArgumentException](#)

`toClamp` is wider and/or taller than this rect.

# Operators

## operator +(IntRect, IntVector2)

Translates the rect by the given vector.

```
public static IntRect operator +(IntRect rect, IntVector2 vector)
```

Parameters

`rect` [IntRect](#)

`vector` [IntVector2](#)

Returns

[IntRect](#)

## operator +(IntVector2, IntRect)

Translates the rect by the given vector.

```
public static IntRect operator +(IntVector2 vector, IntRect rect)
```

Parameters

`vector` [IntVector2](#)

`rect` [IntRect](#)

Returns

[IntRect](#)

## operator ==(IntRect, IntRect)

Whether the two rects describe the same set of points.

```
public static bool operator ==(IntRect a, IntRect b)
```

Parameters

a [IntRect](#)

b [IntRect](#)

Returns

[bool](#)

Remarks

This is just the default [struct](#) comparison (comparing fields).

### See Also

[Equals\(IntRect\)](#), [SetEquals\(IntRect\)](#)

## implicit operator RectInt(IntRect)

Casts to Unity's RectInt.

```
public static implicit operator RectInt(IntRect rect)
```

Parameters

rect [IntRect](#)

Returns

RectInt

## implicit operator IntRect(RectInt)

Casts from Unity's RectInt.

```
public static implicit operator IntRect(RectInt rect)
```

Parameters

rect RectInt

Returns

[IntRect](#)

## operator !=(IntRect, IntRect)

Whether the two rects describe different sets of points.

```
public static bool operator !=(IntRect a, IntRect b)
```

Parameters

a [IntRect](#)

b [IntRect](#)

Returns

[bool](#)

Remarks

This is just the default [struct](#) comparison (comparing fields).

### See Also

[Equals\(IntRect\)](#), [SetEquals\(IntRect\)](#)

## operator -(IntRect, IntVector2)

Translates the rect by the given vector.

```
public static IntRect operator -(IntRect rect, IntVector2 vector)
```

Parameters

rect [IntRect](#)

vector [IntVector2](#)

Returns

[IntRect](#)

## operator -(IntRect)

Negates each point in the rect.

```
public static IntRect operator -(IntRect rect)
```

Parameters

rect [IntRect](#)

Returns

[IntRect](#)

## See Also

[IntVector2](#)

[IntRange](#)

# Struct IntVector2

Namespace: [PAC.DataStructures](#)

A 2-dimensional vector with integer coordinates.

```
public readonly struct IntVector2 : IEquatable<IntVector2>
```

## Implements

[IEquatable](#)<[IntVector2](#)>

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Remarks

Can be implicitly cast to/from ([int](#) `x`, [int](#) `y`). This allows writing [IntVector2](#)s in a more readable way. For example,

```
IntVector2 point = (5, 3);
```

instead of

```
IntVector2 point = new IntVector2(5, 3);
```

## Constructors

### IntVector2(int, int)

Creates a new vector with the given x and y coordinates.

```
public IntVector2(int x, int y)
```

## Parameters

x [int](#)

y [int](#)

## Fields

### down

The vector (0, -1).

```
public static readonly IntVector2 down
```

### Field Value

[IntVector2](#)

### See Also

[right](#), [left](#), [up](#), [upRight](#), [upLeft](#), [downRight](#), [downLeft](#)

### downLeft

The vector (-1, -1).

```
public static readonly IntVector2 downLeft
```

### Field Value

[IntVector2](#)

### See Also

[right](#), [left](#), [up](#), [down](#), [upRight](#), [upLeft](#), [downRight](#)

### downRight

The vector (1, -1).

```
public static readonly IntVector2 downRight
```

Field Value

[IntVector2](#)

**See Also**

[right](#), [left](#), [up](#), [down](#), [upRight](#), [upLeft](#), [downLeft](#)

**left**

The vector (-1, 0).

```
public static readonly IntVector2 left
```

Field Value

[IntVector2](#)

**See Also**

[right](#), [up](#), [down](#), [upRight](#), [upLeft](#), [downRight](#), [downLeft](#)

**one**

The vector (1, 1).

```
public static readonly IntVector2 one
```

Field Value

[IntVector2](#)

**Remarks**

An alias for [upRight](#).

# right

The vector (1, 0).

```
public static readonly IntVector2 right
```

## Field Value

[IntVector2](#)

### See Also

[left](#), [up](#), [down](#), [upRight](#), [upLeft](#), [downRight](#), [downLeft](#)

# up

The vector (0, 1).

```
public static readonly IntVector2 up
```

## Field Value

[IntVector2](#)

### See Also

[right](#), [left](#), [down](#), [upRight](#), [upLeft](#), [downRight](#), [downLeft](#)

# upDownLeftRight

An [IEnumerable<T>](#) over the vectors (0, 1), (0, -1), (-1, 0), (1, 0).

```
public static readonly IEnumerable<IntVector2> upDownLeftRight
```

## Field Value

[IEnumerable](#)<[IntVector2](#)>

### See Also

[up](#), [down](#), [left](#), [right](#)

## upLeft

The vector (-1, 1).

```
public static readonly IntVector2 upLeft
```

Field Value

[IntVector2](#)

### See Also

[right](#), [left](#), [up](#), [down](#), [upRight](#), [downRight](#), [downLeft](#)

## upRight

The vector (1, 1).

```
public static readonly IntVector2 upRight
```

Field Value

[IntVector2](#)

Remarks

An alias for [one](#).

### See Also

[right](#), [left](#), [up](#), [down](#), [upLeft](#), [downRight](#), [downLeft](#)

## X

The x coordinate.

```
public readonly int x
```

Field Value

[int](#)

## See Also

[y](#)

y

The y coordinate.

```
public readonly int y
```

Field Value

[int](#)

## See Also

[x](#)

zero

The vector (0, 0).

```
public static readonly IntVector2 zero
```

Field Value

[IntVector2](#)

# Properties

## |1Norm

The |1 norm of the vector, which is `abs(x) + abs(y)`. Also known as the taxicab/Manhattan norm.

```
public int l1Norm { get; }
```

## Property Value

[int](#)

### See Also

[L1Norm\(IntVector2\)](#), [L1Distance\(IntVector2, IntVector2\)](#),

[https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)#Taxicab\\_norm\\_or\\_Manhattan\\_norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Taxicab_norm_or_Manhattan_norm)

## magnitude

The standard Euclidean magnitude of the vector, which is  $\sqrt{x^2 + y^2}$ .

```
public float magnitude { get; }
```

## Property Value

[float](#)

### See Also

[Magnitude\(IntVector2\)](#), [Distance\(IntVector2, IntVector2\)](#), [sqrMagnitude](#),

[https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)#Euclidean\\_norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Euclidean_norm)

## sign

The sign of each component of the vector.

```
public IntVector2 sign { get; }
```

## Property Value

[IntVector2](#)

## Remarks

The sign of a component that is 0 is defined to be 0.

### See Also

[Sign\(IntVector2\)](#)

# sqrMagnitude

The square of the Euclidean magnitude of the vector, so  $x^2 + y^2$ .

```
public float sqrMagnitude { get; }
```

Property Value

[float](#)

Remarks

This is faster than squaring [magnitude](#).

## See Also

[SqrMagnitude\(IntVector2\)](#), [SqrDistance\(IntVector2, IntVector2\)](#),  
[https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)#Euclidean\\_norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Euclidean_norm)

# supNorm

The supremum norm of the vector, which is  $\max(\text{abs}(x), \text{abs}(y))$ . Also known as the Chebyshev norm or l-infinity norm.

```
public int supNorm { get; }
```

Property Value

[int](#)

## See Also

[SupNorm\(IntVector2\)](#), [SupDistance\(IntVector2, IntVector2\)](#),  
[https://en.wikipedia.org/wiki/Uniform\\_norm](https://en.wikipedia.org/wiki/Uniform_norm)

# Methods

## Abs(IntVector2)

Takes the absolute value component-wise.

```
public static IntVector2 Abs(IntVector2 a)
```

Parameters

a [IntVector2](#)

Returns

[IntVector2](#)

## AreColinear(IntVector2, IntVector2)

Returns whether the two vectors lie on the same real line through (0, 0). This is equivalent to whether there is an [IntVector2](#) dividing both of them.

```
public static bool AreColinear(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[bool](#)

Remarks

For a proof that the two statements given in the summary are logically equivalent, see the source code for this method.

### See Also

[Divides\(IntVector2\)](#)

## ArePerpendicular(IntVector2, IntVector2)

Determines whether the two vectors are perpendicular to each other.

```
public static bool ArePerpendicular(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[bool](#)

Remarks

Being *perpendicular* is also known as being *orthogonal*.

**See Also**

[Dot\(IntVector2, IntVector2\)](#)

## CeilToIntVector2(Vector2)

Ceils the vector component-wise.

```
public static IntVector2 CeilToIntVector2(Vector2 vector2)
```

Parameters

vector2 [Vector2](#)

Returns

[IntVector2](#)

Remarks

Uses `Mathf.CeilToInt(float)`.

**See Also**

[FloorToIntVector2\(Vector2\)](#), [RoundToIntVector2\(Vector2\)](#)

## Deconstruct(out int, out int)

Deconstructs the vector into its x and y coordinates.

```
public void Deconstruct(out int x, out int y)
```

Parameters

x [int](#)

y [int](#)

## Distance(IntVector2, IntVector2)

Computes the standard Euclidean distance between the two vectors - i.e. the [Magnitude\(IntVector2\)](#) of  $a - b$ .

```
public static float Distance(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[float](#)

### See Also

[SqrDistance\(IntVector2, IntVector2\)](#)

## Divides(IntVector2)

Returns whether [dividend](#) is an integer multiple of this vector.

For example,  $(1, 2)$  divides  $(3, 6)$  as  $(3, 6) = 3 * (1, 2)$ .

```
public bool Divides(IntVector2 dividend)
```

Parameters

`dividend` [IntVector2](#)

Returns

`bool` ↗

Remarks

More precisely, this determines whether there exists an integer `n` such that `dividend = n * this`. This means, for example, that `(0, 3)` divides `(0, 6)`, and that everything divides `(0, 0)`.

### See Also

[IsMultipleOf\(IntVector2\)](#)

## Dot(IntVector2, IntVector2)

Computes the dot product of the two vectors.

```
public static int Dot(IntVector2 a, IntVector2 b)
```

Parameters

`a` [IntVector2](#)

`b` [IntVector2](#)

Returns

`int` ↗

## Equals(IntVector2)

The same as [operator ==\(IntVector2, IntVector2\)](#)

```
public bool Equals(IntVector2 other)
```

Parameters

other [IntVector2](#)

Returns

[bool](#)

## Equals(object)

The same as [Equals\(IntVector2\)](#).

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns the vector flipped across the given axis.

```
public IntVector2 Flip(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

Returns

[IntVector2](#)

## FloorToIntVector2(Vector2)

Floors the vector component-wise.

```
public static IntVector2 FloorToIntVector2(Vector2 vector2)
```

Parameters

`vector2` Vector2

Returns

[IntVector2](#)

Remarks

Uses Mathf.FloorToInt(float).

### See Also

[CeilToIntVector2\(Vector2\)](#), [RoundToIntVector2\(Vector2\)](#)

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## IsMultipleOf(IntVector2)

Returns whether this vector is an integer multiple of `divisor`.

For example,  $(3, 6)$  is a multiple of  $(1, 2)$  as  $(3, 6) = 3 * (1, 2)$ .

```
public bool IsMultipleOf(IntVector2 divisor)
```

Parameters

`divisor` [IntVector2](#)

Returns

`bool` ↗

Remarks

More precisely, this determines whether there exists an integer `n` such that `this = n * divisor`. This means, for example, that  $(0, 6)$  is a multiple of  $(0, 3)$ , and that  $(0, 0)$  is a multiple of everything.

**See Also**

[Divides\(IntVector2\)](#)

## L1Distance(IntVector2, IntVector2)

Computes the L1 distance between the two vectors - i.e. the [L1Norm\(IntVector2\)](#) of  $a - b$ . Also known as the taxicab/Manhattan distance or rectilinear distance.

```
public static int L1Distance(IntVector2 a, IntVector2 b)
```

Parameters

`a` [IntVector2](#)

`b` [IntVector2](#)

Returns

`int` ↗

## L1Norm(IntVector2)

The l1 norm of the vector, which is `abs(x) + abs(y)`. Also known as the taxicab/Manhattan norm.

```
public static int L1Norm(IntVector2 a)
```

Parameters

a [IntVector2](#)

Returns

[int](#)

**See Also**

[L1Norm](#), [L1Distance\(IntVector2, IntVector2\)](#),

[https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)#Taxicab\\_norm\\_or\\_Manhattan\\_norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Taxicab_norm_or_Manhattan_norm)

## Magnitude(IntVector2)

The standard Euclidean magnitude of the vector, which is `sqrt(x^2 + y^2)`.

```
public static float Magnitude(IntVector2 a)
```

Parameters

a [IntVector2](#)

Returns

[float](#)

**See Also**

[magnitude](#), [Distance\(IntVector2, IntVector2\)](#), [SqrMagnitude\(IntVector2\)](#),

[https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)#Euclidean\\_norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Euclidean_norm)

## Max(IntVector2, IntVector2)

Takes the maximum of the two vectors component-wise.

```
public static IntVector2 Max(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[IntVector2](#)

**See Also**

[Min\(IntVector2, IntVector2\)](#)

## Max(params IntVector2[])

Takes the maximum of the vectors component-wise.

```
public static IntVector2 Max(params IntVector2[] vectors)
```

Parameters

vectors [IntVector2\[\]](#)

Returns

[IntVector2](#)

Exceptions

[ArgumentNullException](#)

vectors is null.

[ArgumentException](#)

vectors is empty.

**See Also**

[Min\(params IntVector2\[\]\)](#)

## Max(IEnumerable<IntVector2>)

Takes the maximum of the vectors component-wise.

```
public static IntVector2 Max(IEnumerable<IntVector2> vectors)
```

### Parameters

vectors [IEnumerable<IntVector2>](#)

### Returns

[IntVector2](#)

### Remarks

This is not provided as an [IEnumerable<T>](#) extension method since it seems C# will favour using the LINQ [Max<TSource>\(IEnumerable<TSource>\)](#) instead, which causes errors.

### Exceptions

[ArgumentNullException](#)

vectors is null.

[ArgumentException](#)

vectors is empty.

### See Also

[Min\(IEnumerable<IntVector2>\)](#)

## Min(IntVector2, IntVector2)

Takes the minimum of the two vectors component-wise.

```
public static IntVector2 Min(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[IntVector2](#)

**See Also**

[Max\(IntVector2, IntVector2\)](#)

## Min(params IntVector2[])

Takes the minimum of the vectors component-wise.

```
public static IntVector2 Min(params IntVector2[] vectors)
```

Parameters

vectors [IntVector2\[\]](#)

Returns

[IntVector2](#)

Exceptions

[ArgumentNullException](#)

vectors is null.

[ArgumentException](#)

vectors is empty.

**See Also**

[Max\(params IntVector2\[\]\).](#)

## Min(IEnumerable<IntVector2>)

Takes the minimum of the vectors component-wise.

```
public static IntVector2 Min(IEnumerable<IntVector2> vectors)
```

### Parameters

vectors [IEnumerable<IntVector2>](#)

### Returns

[IntVector2](#)

### Remarks

This is not provided as an [IEnumerable<T>](#) extension method since it seems C# will favour using the LINQ [Min<TSource>\(IEnumerable<TSource>\)](#) instead, which causes errors.

### Exceptions

[ArgumentNullException](#)

vectors is null.

[ArgumentException](#)

vectors is empty.

### See Also

[Max\(IEnumerable<IntVector2>\).](#)

## Rotate(RotationAngle)

Returns the vector rotated clockwise by the given angle.

```
public IntVector2 Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

[IntVector2](#)

## RoundToIntVector2(Vector2)

Rounds the vector component-wise.

```
public static IntVector2 RoundToIntVector2(Vector2 vector2)
```

Parameters

vector2 [Vector2](#)

Returns

[IntVector2](#)

Remarks

Uses Mathf.RoundToInt(float).

### See Also

[FloorToIntVector2\(Vector2\)](#), [CeilToIntVector2\(Vector2\)](#).

## Sign(IntVector2)

Returns the sign of each component of the vector.

```
public static IntVector2 Sign(IntVector2 a)
```

Parameters

a [IntVector2](#)

Returns

[IntVector2](#)

Remarks

The sign of a component that is 0 is defined to be 0.

**See Also**

[sign](#)

## Simplify(IntVector2)

Scales the vector down so its components are coprime (have no common divisors). In other words, it computes the smallest [IntVector2](#) dividing a.

```
public static IntVector2 Simplify(IntVector2 a)
```

Parameters

a [IntVector2](#)

Returns

[IntVector2](#)

Remarks

Preserves signs.

## SqrDistance(IntVector2, IntVector2)

Computes the square of the Euclidean distance between the two vectors - i.e. the [Sqr Magnitude\(IntVector2\)](#) of  $a - b$ .

```
public static float SqrDistance(IntVector2 a, IntVector2 b)
```

## Parameters

a [IntVector2](#)

b [IntVector2](#)

## Returns

[float](#)

## Remarks

This is faster than squaring [Distance\(IntVector2, IntVector2\)](#).

## SqrMagnitude(IntVector2)

The square of the Euclidean magnitude of the vector, so  $x^2 + y^2$ .

```
public static float SqrMagnitude(IntVector2 a)
```

## Parameters

a [IntVector2](#)

## Returns

[float](#)

## Remarks

This is faster than squaring [Magnitude\(IntVector2\)](#).

## See Also

[sqrMagnitude](#), [SqrDistance\(IntVector2, IntVector2\)](#),

[https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)#Euclidean\\_norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Euclidean_norm)

## SupDistance(IntVector2, IntVector2)

Computes the supremum distance between the two vectors - i.e. the [SupNorm\(IntVector2\)](#) of  $a - b$ . Also known as the Chebyshev distance or l-infinity distance.

```
public static int SupDistance(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[int](#)

## SupNorm(IntVector2)

The supremum norm of the vector, which is  $\max(\text{abs}(x), \text{abs}(y))$ . Also known as the Chebyshev norm or l-infinity norm.

```
public static int SupNorm(IntVector2 a)
```

Parameters

a [IntVector2](#)

Returns

[int](#)

### See Also

[supNorm](#), [SupDistance\(IntVector2, IntVector2\)](#), [https://en.wikipedia.org/wiki/Uniform\\_norm](https://en.wikipedia.org/wiki/Uniform_norm)

## ToString()

Represents the vector as a string in the form " $(x, y)$ ".

```
public override string ToString()
```

Returns

[string](#)

## Operators

### operator +(IntVector2, IntVector2)

Adds the two vectors component-wise.

```
public static IntVector2 operator +(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[IntVector2](#)

### operator /(IntVector2, IntVector2)

Divides (integer division) the two vectors component-wise.

```
public static IntVector2 operator /(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

## [IntVector2](#)

### operator /(IntVector2, int)

Divides (integer division) each component of the vector by `scalar`.

```
public static IntVector2 operator /(IntVector2 vector, int scalar)
```

Parameters

`vector` [IntVector2](#)

`scalar` [int](#)

Returns

[IntVector2](#)

### operator ==(IntVector2, IntVector2)

Whether the two vectors have identical x and y coords.

```
public static bool operator ==(IntVector2 a, IntVector2 b)
```

Parameters

`a` [IntVector2](#)

`b` [IntVector2](#)

Returns

[bool](#)

#### See Also

[Equals\(IntVector2\)](#)

## explicit operator IntVector2(Vector2)

Casts from Unity's Vector2, by casting each coordinate to [int](#), which truncates them towards zero.

```
public static explicit operator IntVector2(Vector2 vector)
```

Parameters

**vector** Vector2

Returns

[IntVector2](#)

## explicit operator IntVector2(Vector3)

Casts from Unity's Vector3, by casting each coordinate to [int](#), which truncates them towards zero. The z coordinate is ignored.

```
public static explicit operator IntVector2(Vector3 vector)
```

Parameters

**vector** Vector3

Returns

[IntVector2](#)

## operator >(IntVector2, IntVector2)

Returns true iff > holds for both components.

```
public static bool operator >(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[bool](#)

## operator >=(IntVector2, IntVector2)

Returns true iff  $\geq$  holds for both components.

```
public static bool operator >=(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[bool](#)

## implicit operator (int x, int y)(IntVector2)

```
public static implicit operator (int x, int y)(IntVector2 vector)
```

Parameters

vector [IntVector2](#)

Returns

([int](#) x, [int](#) y)

Remarks

This conversion should be inlined by the JIT compiler.

## implicit operator Vector2(IntVector2)

Casts to Unity's Vector2, by casting each coordinate to [float](#).

```
public static implicit operator Vector2(IntVector2 vector)
```

Parameters

vector [IntVector2](#)

Returns

Vector2

## implicit operator Vector2Int(IntVector2)

Casts to Unity's Vector2Int.

```
public static implicit operator Vector2Int(IntVector2 vector)
```

Parameters

vector [IntVector2](#)

Returns

Vector2Int

## implicit operator Vector3(IntVector2)

Casts to Unity's Vector3, by casting each coordinate to [float](#), with a 0 in the z-coord.

```
public static implicit operator Vector3(IntVector2 vector)
```

Parameters

vector [IntVector2](#)

Returns

Vector3

## implicit operator IntVector2((int x, int y))

Allows writing [IntVector2](#)s in a more readable way

For example,

```
IntVector2 point = (5, 3);
```

instead of

```
IntVector2 point = new IntVector2(5, 3);
```

```
public static implicit operator IntVector2((int x, int y) tuple)
```

Parameters

**tuple** ([int](#) [x](#), [int](#) [y](#))

Returns

[IntVector2](#)

Remarks

This conversion should be inlined by the JIT compiler.

## implicit operator IntVector2(Vector2Int)

Casts from Unity's Vector2Int.

```
public static implicit operator IntVector2(Vector2Int vector)
```

Parameters

**vector** Vector2Int

Returns

[IntVector2](#)

## operator !=(IntVector2, IntVector2)

Whether the two vectors differ in their x and/or y coords.

```
public static bool operator !=(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[bool](#)

### See Also

[Equals\(IntVector2\)](#)

## operator <(IntVector2, IntVector2)

Returns true iff < holds for both components.

```
public static bool operator <(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[bool](#)

## operator <=(IntVector2, IntVector2)

Returns true iff  $\leq$  holds for both components.

```
public static bool operator <=(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[bool](#) ↗

## operator \*(IntVector2, IntVector2)

Multiplies the two vectors component-wise.

```
public static IntVector2 operator *(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[IntVector2](#)

## operator \*(IntVector2, int)

Multiplies each component of the vector by [scalar](#).

```
public static IntVector2 operator *(IntVector2 vector, int scalar)
```

Parameters

vector [IntVector2](#)

scalar [int](#)

Returns

[IntVector2](#)

## operator \*(int, IntVector2)

Multiplies each component of the vector by `scalar`.

```
public static IntVector2 operator *(int scalar, IntVector2 vector)
```

Parameters

scalar [int](#)

vector [IntVector2](#)

Returns

[IntVector2](#)

## operator -(IntVector2, IntVector2)

Subtracts the two vectors component-wise.

```
public static IntVector2 operator -(IntVector2 a, IntVector2 b)
```

Parameters

a [IntVector2](#)

b [IntVector2](#)

Returns

[IntVector2](#)

## operator -(IntVector2)

Negates each component of the vector.

```
public static IntVector2 operator -(IntVector2 a)
```

Parameters

a [IntVector2](#)

Returns

[IntVector2](#)

# Struct Rational

Namespace: [PAC.DataStructures](#)

A fraction  $a / b$  (where  $a$  and  $b$  are integers) kept in simplest form.

```
public readonly struct Rational : IEquatable<Rational>, IEquatable<int>
```

## Implements

[IEquatable](#)<[Rational](#)>, [IEquatable](#)<[int](#)>

## Inherited Members

[object.Equals\(object, object\)](#), [object.GetType\(\)](#),  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#),  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Remarks

Note that small values can have large [numerators](#) and [denominators](#), which could cause overflow in e.g. [operator +\(Rational, Rational\)](#).

## Constructors

### Rational(int, int)

```
public Rational(int numerator, int denominator)
```

#### Parameters

numerator [int](#)

See [numerator](#).

denominator [int](#)

See [denominator](#).

## Remarks

Will simplify the fraction.

## Exceptions

### [ArgumentOutOfRangeException](#)

`numerator` or `denominator` are [MinValue](#).

## Fields

### Epsilon

The smallest positive value that can be represented.

```
public static readonly Rational Epsilon
```

### Field Value

#### [Rational](#)

### Half

The value 1/2.

```
public static readonly Rational Half
```

### Field Value

#### [Rational](#)

### MaxValue

The most positive value that can be represented.

```
public static readonly Rational MaxValue
```

## Field Value

[Rational](#)

### See Also

[MinValue](#)

## MinValue

The most negative value that can be represented.

```
public static readonly Rational MinValue
```

## Field Value

[Rational](#)

### See Also

[MaxValue](#)

## Undefined

Represents any value with denominator 0, but is stored as 0/0.

```
public static readonly Rational Undefined
```

## Field Value

[Rational](#)

## Remarks

Note that this will not be considered == to itself. Use [isUndefined](#) instead to determine if a [Rational](#) is [Undefined](#).

## denominator

The bottom number of the fraction.

```
public readonly int denominator
```

## Field Value

[int](#)

## Remarks

This will always be non-negative and coprime to the [numerator](#).

If the [numerator](#) is 0 and this is not 0, this will be 1.

## numerator

The top number of the fraction.

```
public readonly int numerator
```

## Field Value

[int](#)

## Remarks

This will always be coprime to the [denominator](#) and its sign will always be the sign of the [Rational](#).

If the [denominator](#) is 0, this will be 0 (and this [Rational](#) represents an undefined value).

## See Also

[denominator](#)

## Properties

### isInteger

Whether the [denominator](#) is 1.

```
public bool isInteger { get; }
```

Property Value

[bool](#) ↗

## isUndefined

Whether the [Rational](#) is [Undefined](#).

```
public bool isUndefined { get; }
```

Property Value

[bool](#) ↗

## Remarks

Note that two [Undefined](#) values are not considered ==, so use this method to check if a [Rational](#) is [Undefined](#).

## sign

The sign of the [Rational](#).

```
public int sign { get; }
```

Property Value

[int](#) ↗

- 1 if the value is positive
- 0 if the value is zero or [Undefined](#)
- -1 if the value is negative

## Remarks

This will match the sign of the [numerator](#).

# Methods

## Abs(Rational)

Returns the absolute value of [a](#).

```
public static Rational Abs(Rational a)
```

### Parameters

[a](#) [Rational](#)

### Returns

[Rational](#)

### Remarks

If [a](#) is [Undefined](#), it will return [Undefined](#).

## Equals(Rational)

See [operator ==\(Rational, Rational\)](#).

```
public bool Equals(Rational other)
```

### Parameters

[other](#) [Rational](#)

### Returns

[bool](#) ↗

## Equals(int)

See [implicit operator Rational\(int\)](#) and [operator ==\(Rational, Rational\)](#).

```
public bool Equals(int other)
```

Parameters

other [int](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(Rational\)](#).

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

Returns

[bool](#)

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

[string](#)

The fully qualified type name.

## Operators

### operator +(Rational, Rational)

```
public static Rational operator +(Rational a, Rational b)
```

Parameters

a [Rational](#)

b [Rational](#)

Returns

[Rational](#)

Remarks

If either of a and b are [Undefined](#), it will return [Undefined](#).

### operator /(Rational, Rational)

```
public static Rational operator /(Rational a, Rational b)
```

Parameters

a [Rational](#)

b [Rational](#)

Returns

[Rational](#)

Remarks

If either of a and b are [Undefined](#), or if b is 0, it will return [Undefined](#).

## operator ==(Rational, Rational)

Whether the two [Rationals](#) represent the same fraction.

```
public static bool operator ==(Rational a, Rational b)
```

Parameters

a [Rational](#)

b [Rational](#)

Returns

[bool](#)

Remarks

Two [Undefined](#) values are not considered ==. Use [isUndefined](#) to check if a [Rational](#) is [Undefined](#).

## explicit operator int(Rational)

Rounds the [Rational](#) towards zero.

```
public static explicit operator int(Rational rational)
```

Parameters

`rational` [Rational](#)

Returns

[int](#)

Exceptions

[DivideByZeroException](#)

`rational` is [Undefined](#).

## operator >(Rational, Rational)

Whether `a` > `b`.

```
public static bool operator >(Rational a, Rational b)
```

Parameters

`a` [Rational](#)

`b` [Rational](#)

Returns

[bool](#)

Remarks

Returns [false](#) if either of `a` or `b` are [Undefined](#).

## operator >=(Rational, Rational)

Whether `a` >= `b`.

```
public static bool operator >=(Rational a, Rational b)
```

Parameters

a [Rational](#)

b [Rational](#)

Returns

[bool](#)

Remarks

Returns [false](#) if either of a or b are [Undefined](#).

## implicit operator float(Rational)

Converts the [Rational](#) to a [float](#).

```
public static implicit operator float(Rational rational)
```

Parameters

rational [Rational](#)

Returns

[float](#)

Remarks

[Undefined](#) will be converted to [NaN](#).

## implicit operator Rational(int)

Converts the integer into a [Rational](#) with [denominator](#) 1.

```
public static implicit operator Rational(int integer)
```

Parameters

integer [int](#)

Returns

[Rational](#)

## operator !=(Rational, Rational)

See [operator ==\(Rational, Rational\)](#).

```
public static bool operator !=(Rational a, Rational b)
```

Parameters

a [Rational](#)

b [Rational](#)

Returns

[bool](#)

## operator <(Rational, Rational)

Whether a < b.

```
public static bool operator <(Rational a, Rational b)
```

Parameters

a [Rational](#)

b [Rational](#)

Returns

[bool](#)

## Remarks

Returns [false](#) if either of **a** or **b** are [Undefined](#).

## operator <=(Rational, Rational)

Whether **a** <= **b**.

```
public static bool operator <=(Rational a, Rational b)
```

## Parameters

**a** [Rational](#)

**b** [Rational](#)

## Returns

[bool](#)

## Remarks

Returns [false](#) if either of **a** or **b** are [Undefined](#).

## operator %(Rational, Rational)

Returns the smallest non-negative [Rational](#) **r** such that there is an integer **q** such that **a** = **q** \* **b** + **r**.

```
public static Rational operator %(Rational a, Rational b)
```

## Parameters

**a** [Rational](#)

**b** [Rational](#)

## Returns

## [Rational](#)

### Remarks

If either of `a` and `b` are [Undefined](#), it will return [Undefined](#).

## operator \*(Rational, Rational)

```
public static Rational operator *(Rational a, Rational b)
```

### Parameters

`a` [Rational](#)

`b` [Rational](#)

### Returns

[Rational](#)

### Remarks

If either of `a` and `b` are [Undefined](#), it will return [Undefined](#).

## operator -(Rational, Rational)

```
public static Rational operator -(Rational a, Rational b)
```

### Parameters

`a` [Rational](#)

`b` [Rational](#)

### Returns

[Rational](#)

## Remarks

If either of `a` and `b` are [Undefined](#), it will return [Undefined](#).

## operator -(Rational)

```
public static Rational operator -(Rational a)
```

## Parameters

`a` [Rational](#)

## Returns

[Rational](#)

## Remarks

If `a` is [Undefined](#), it will return [Undefined](#).

# Struct SemanticVersion

Namespace: [PAC.DataStructures](#)

```
public struct SemanticVersion
```

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Constructors

### SemanticVersion(int, int, int)

```
public SemanticVersion(int major, int minor, int patch)
```

## Parameters

major [int](#)

minor [int](#)

patch [int](#)

## Properties

### major

```
public int major { get; set; }
```

## Property Value

[int](#)

## minor

```
public int minor { get; set; }
```

Property Value

[int](#)

## patch

```
public int patch { get; set; }
```

Property Value

[int](#)

## Methods

### Equals(object)

Indicates whether this instance and a specified object are equal.

```
public override bool Equals(object obj)
```

Parameters

**obj** [object](#)

The object to compare with the current instance.

Returns

[bool](#)

[true](#) if `obj` and this instance are the same type and represent the same value; otherwise, [false](#).

## GetHashCode()

Returns the hash code for this instance.

```
public override int GetHashCode()
```

Returns

[int](#)

A 32-bit signed integer that is the hash code for this instance.

## IncrementMajor()

Adds 1 to the major and resets the minor and patch to 0.

```
public void IncrementMajor()
```

## IncrementMinor()

Adds 1 to the minor and resets the patch to 0. (The major is unchanged.)

```
public void IncrementMinor()
```

## IncrementPatch()

Adds 1 to the patch. (The major and minor are unchanged.)

```
public void IncrementPatch()
```

## ToString()

Returns the fully qualified type name of this instance.

```
public override string ToString()
```

Returns

[string](#)

The fully qualified type name.

## Operators

### operator ==(SemanticVersion, SemanticVersion)

```
public static bool operator ==(SemanticVersion version1, SemanticVersion version2)
```

Parameters

version1 [SemanticVersion](#)

version2 [SemanticVersion](#)

Returns

[bool](#)

### operator >(SemanticVersion, SemanticVersion)

```
public static bool operator >(SemanticVersion version1, SemanticVersion version2)
```

Parameters

version1 [SemanticVersion](#)

version2 [SemanticVersion](#)

Returns

[bool](#)

## operator >=(SemanticVersion, SemanticVersion)

```
public static bool operator >=(SemanticVersion version1, SemanticVersion version2)
```

Parameters

[version1](#) [SemanticVersion](#)

[version2](#) [SemanticVersion](#)

Returns

[bool](#)

## operator !=(SemanticVersion, SemanticVersion)

```
public static bool operator !=(SemanticVersion version1, SemanticVersion version2)
```

Parameters

[version1](#) [SemanticVersion](#)

[version2](#) [SemanticVersion](#)

Returns

[bool](#)

## operator <(SemanticVersion, SemanticVersion)

```
public static bool operator <(SemanticVersion version1, SemanticVersion version2)
```

Parameters

`version1 SemanticVersion`

`version2 SemanticVersion`

Returns

`bool` ↗

## operator <=(SemanticVersion, SemanticVersion)

```
public static bool operator <=(SemanticVersion version1, SemanticVersion version2)
```

Parameters

`version1 SemanticVersion`

`version2 SemanticVersion`

Returns

`bool` ↗

## operator -(SemanticVersion, SemanticVersion)

Returns a Version such that the most important field in which the operands differ (in the order major, minor, patch) is set to the positive difference of them in this field, and the other two fields are set to 0.

NOTE: This operation is commutative (swapping the order of the operands doesn't affect the result).

Example:

```
Version(1, 4, 3) - Version(1, 6, 10) = Version(0, 2, 0)
```

```
public static SemanticVersion operator -(SemanticVersion version1,  
SemanticVersion version2)
```

## Parameters

`version1` [SemanticVersion](#)

`version2` [SemanticVersion](#)

## Returns

[SemanticVersion](#)

# Class SemanticVersion.JsonConverter

Namespace: [PAC.DataStructures](#)

```
public class SemanticVersion.JsonConverter :  
JsonConversion.JsonConverter<SemanticVersion, JsonData.String>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<SemanticVersion, JsonData.String>](#) ←  
SemanticVersion.JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<SemanticVersion, JsonData.String>.ToJson\(SemanticVersion\)](#),  
,

[JsonConversion.JsonConverter<SemanticVersion, JsonData.String>.FromJson\(JsonData.String\)](#),  
[JsonConversion.JsonConverter<SemanticVersion, JsonData.String>.FromJson\(JsonData\)](#),  
[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#),  
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#),  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#),  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### FromJson(String)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override SemanticVersion FromJson(JsonData.String jsonData)
```

## Parameters

`jsonData` [JsonData.String](#)

Returns

[SemanticVersion](#)

## ToJson(SemanticVersion)

Attempts to convert the C# object into JSON data.

```
public override JsonData.String ToJson(SemanticVersion version)
```

Parameters

`version` [SemanticVersion](#)

Returns

[JsonData.String](#)

# Namespace PAC.Drawing

## Classes

### [CoordSnapping](#)

Provides methods to snap coordinates so they fit certain properties, such as forming a square.

### [DrawingArea](#)

### [GridManager](#)

Handles drawing a grid over the image when the grid is enabled.

### [Tool](#)

Defines the tools available to use and their properties.

### [Toolbar](#)

Handles selecting tools, brush size, etc.

### [Tools](#)

Defines how different tools act. I may rework this to be more like the BlendMode class where each Tool instance defines how it acts. Then to use a tool you would just do tool.Use(pixel) or similar.

## Enums

### [BrushShape](#)

### [GradientMode](#)

### [SelectionMode](#)

### [Shape](#)

# Enum BrushShape

Namespace: [PAC.Drawing](#)

```
public enum BrushShape
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Circle = 0

Custom = -1

Diamond = 2

Square = 1

# Class CoordSnapping

Namespace: [PAC.Drawing](#)

Provides methods to snap coordinates so they fit certain properties, such as forming a square.

```
public static class CoordSnapping
```

## Inheritance

[object](#) ← CoordSnapping

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### SnapToSquare(IntVector2, IntVector2)

Either changes `movablePoint`'s x coord or changes its y coord so that the rectangle it forms with `fixedPoint` is a square.

```
public static IntVector2 SnapToSquare(IntVector2 fixedPoint,  
IntVector2 movablePoint)
```

#### Parameters

`fixedPoint` [IntVector2](#)

`movablePoint` [IntVector2](#)

#### Returns

[IntVector2](#)

#### Remarks

Chooses the largest such square that preserves the sign of each component of `movablePoint` - `fixedPoint`. If `fixedPoint` and `movablePoint` have the same x coord (but different y), the square will be made by increasing the x coord; if `fixedPoint` and `movablePoint` have the same y coord (but different x), the square will be made by increasing the y coord.

# Class DrawingArea

Namespace: [PAC.Drawing](#)

```
public class DrawingArea : MonoBehaviour
```

## Inheritance

[object](#) ← DrawingArea

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### file

```
public File file { get; }
```

#### Property Value

[File](#)

### hasSelection

```
public bool hasSelection { get; }
```

#### Property Value

[bool](#)

### pixelsPerUnit

```
public float pixelsPerUnit { get; }
```

Property Value

[float](#)

## scrollSpeed

```
public float scrollSpeed { get; }
```

Property Value

[float](#)

## selectionMask

```
public Texture2D selectionMask { get; }
```

Property Value

Texture2D

## selectionRect

```
public IntRect selectionRect { get; }
```

Property Value

[IntRect](#)

## zoomScrollSpeed

```
public float zoomScrollSpeed { get; }
```

Property Value

## Methods

### DeleteSelection()

```
public void DeleteSelection()
```

### PixelsToWorldPos(IntVector2)

Turns the pixel coordinate in the drawing into a world coordinate (the resulting coord is the centre of the pixel in the world).

```
public Vector2 PixelsToWorldPos(IntVector2 pixel)
```

#### Parameters

`pixel` [IntVector2](#)

#### Returns

`Vector2`

### PixelsToWorldPos(int, int)

Turns the pixel coordinate in the drawing into a world coordinate (the resulting coord is the centre of the pixel in the world).

```
public Vector2 PixelsToWorldPos(int x, int y)
```

#### Parameters

`x` [int](#)

`y` [int](#)

Returns

Vector2

## PixelsToWorldPos(float, float)

Turns the pixel coordinate in the drawing into a world coordinate.

```
public Vector2 PixelsToWorldPos(float x, float y)
```

Parameters

x [float](#)

y [float](#)

Returns

Vector2

## PixelsToWorldPos(Vector2)

Turns the pixel coordinate in the drawing into a world coordinate.

```
public Vector2 PixelsToWorldPos(Vector2 pixel)
```

Parameters

pixel Vector2

Returns

Vector2

## UpdateDrawing()

Makes any changes to the file visible on the drawing area.

```
public void UpdateDrawing()
```

## WorldPosToPixel(Vector2)

Turns the world coord into a pixel coord in the drawing.

```
public IntVector2 WorldPosToPixel(Vector2 worldPos)
```

### Parameters

`worldPos Vector2`

### Returns

[IntVector2](#)

## WorldPosToPixels(float, float)

Turns the world coord into a pixel coord in the drawing.

```
public IntVector2 WorldPosToPixels(float x, float y)
```

### Parameters

`x float`

`y float`

### Returns

[IntVector2](#)

# Enum GradientMode

Namespace: [PAC.Drawing](#)

```
public enum GradientMode
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Linear = 0

Radial = 1

# Class GridManager

Namespace: [PAC.Drawing](#)

Handles drawing a grid over the image when the grid is enabled.

```
public class GridManager : MonoBehaviour
```

## Inheritance

[object](#) ← GridManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Fields

### lineThickness

```
public float lineThickness
```

Field Value

[float](#)

## Properties

### height

```
public int height { get; }
```

Property Value

[int](#)

## on

```
public bool on { get; }
```

## Property Value

[bool](#)

## width

```
public int width { get; }
```

## Property Value

[int](#)

## xOffset

```
public int xOffset { get; }
```

## Property Value

[int](#)

## yOffset

```
public int yOffset { get; }
```

## Property Value

[int](#)

## Methods

## SetGrid(int, int, int, int)

```
public void SetGrid(int width, int height, int xoffset, int yoffset)
```

### Parameters

width [int](#)

height [int](#)

xOffset [int](#)

yOffset [int](#)

## SetOnOff(bool)

```
public void SetOnOff(bool on)
```

### Parameters

on [bool](#)

## SetOnOffNoDisplayUpdate(bool)

```
public void SetOnOffNoDisplayUpdate(bool on)
```

### Parameters

on [bool](#)

# Enum SelectionMode

Namespace: [PAC.Drawing](#)

```
public enum SelectionMode
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Draw = 0

Ellipse = 11

MagicWand = 1

Rectangle = 10

# Enum Shape

Namespace: [PAC.Drawing](#)

```
public enum Shape
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Diamond = 3

Ellipse = 1

Rectangle = 0

RightTriangle = 2

# Class Tool

Namespace: [PAC.Drawing](#)

Defines the tools available to use and their properties.

```
public class Tool
```

## Inheritance

[object](#) ← Tool

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### Brush

```
public static Tool Brush
```

### Field Value

[Tool](#)

### EyeDropper

```
public static Tool EyeDropper
```

### Field Value

[Tool](#)

## Fill

```
public static Tool Fill
```

Field Value

[Tool](#)

## GlobalEyeDropper

```
public static Tool GlobalEyeDropper
```

Field Value

[Tool](#)

## Gradient

```
public static Tool Gradient
```

Field Value

[Tool](#)

## IsoBox

```
public static Tool IsoBox
```

Field Value

[Tool](#)

## Line

```
public static Tool Line
```

### Field Value

[Tool](#)

## Move

```
public static Tool Move
```

### Field Value

[Tool](#)

## None

```
public static Tool None
```

### Field Value

[Tool](#)

## Pencil

```
public static Tool Pencil
```

### Field Value

[Tool](#)

## Rubber

```
public static Tool Rubber
```

Field Value

[Tool](#)

## Selection

```
public static Tool Selection
```

Field Value

[Tool](#)

## Shape

```
public static Tool Shape
```

Field Value

[Tool](#)

## canBeCancelled

```
public bool canBeCancelled
```

Field Value

[bool](#) ↗

## tools

All implemented tools.

```
public static readonly Tool[] tools
```

Field Value

[Tool\[\]](#)

## Properties

### finishMode

What action causes a use of the tool to be ended.

```
public MouseTargetDeselectMode finishMode { get; }
```

Property Value

[MouseTargetDeselectMode](#)

### name

```
public string name { get; }
```

Property Value

[string](#) ↗

### showBrushBorder

Whether the outline of the brush shape should be shown.

```
public bool showBrushBorder { get; }
```

Property Value

[bool](#) ↗

## useMovementInterpolation

When the mouse position jumps between frames: true - the tool should be applied to each coord the mouse moved through; false - just applied to the ending coord.

```
public bool useMovementInterpolation { get; }
```

Property Value

[bool](#)

## Methods

### StringToTool(string)

Gets the tool with that name.

```
public static Tool StringToTool(string toolName)
```

Parameters

toolName [string](#)

Returns

[Tool](#)

# Class Toolbar

Namespace: [PAC.Drawing](#)

Handles selecting tools, brush size, etc.

```
public class Toolbar : MonoBehaviour
```

## Inheritance

[object](#) ← Toolbar

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### brushPixels

The pixels, given relative to the position of the mouse, that will be affected by the current brush.

```
public IntVector2[] brushPixels { get; }
```

Property Value

[IntVector2\[\]](#)

### brushPixelsHeight

```
public int brushPixelsHeight { get; }
```

Property Value

[int](#)

## brushPixelsIsEmpty

```
public bool brushPixelsIsEmpty { get; }
```

Property Value

[bool](#)

## brushPixelsIsSingleCentralPixel

```
public bool brushPixelsIsSingleCentralPixel { get; }
```

Property Value

[bool](#)

## brushPixelsWidth

```
public int brushPixelsWidth { get; }
```

Property Value

[int](#)

## brushShape

```
public BrushShape brushShape { get; set; }
```

Property Value

[BrushShape](#)

## brushSize

```
public int brushSize { get; }
```

Property Value

[int](#)

## brushTexture

```
public Texture2D brushTexture { get; }
```

Property Value

Texture2D

## gradientMode

```
public GradientMode gradientMode { get; }
```

Property Value

[GradientMode](#)

## lineSmoothingTime

The amount of time you have to draw a new pixel in for an old one to be potentially smoothed.

```
public float lineSmoothingTime { get; }
```

Property Value

[float](#)

## previousTool

```
public Tool previousTool { get; }
```

Property Value

[Tool](#)

## selectedTool

```
public Tool selectedTool { get; }
```

Property Value

[Tool](#)

## selectionMode

```
public SelectionMode selectionMode { get; }
```

Property Value

[SelectionMode](#)

## shapeToolShape

```
public Shape shapeToolShape { get; }
```

Property Value

[Shape](#)

## Methods

## DeselectGlobalEyeDropper()

```
public void DeselectGlobalEyeDropper()
```

## LoadCustomBrush(Texture2D)

Turns the given texture into a brush shape by taking any pixels with non-zero alpha.

```
public void LoadCustomBrush(Texture2D brushShape)
```

### Parameters

brushShape `Texture2D`

## SelectGlobalEyeDropper()

```
public void SelectGlobalEyeDropper()
```

## SetBrushSize(int)

```
public bool SetBrushSize(int brushSize)
```

### Parameters

brushSize `int`

### Returns

`bool`

## SubscribeToOnBrushPixelsChanged(UnityAction)

Event invoked when brush pixels change.

```
public void SubscribeToOnBrushPixelsChanged(UnityAction call)
```

Parameters

**call** UnityAction

## SubscribeToOnBrushSizeChanged(UnityAction)

Event invoked when brush size changes.

```
public void SubscribeToOnBrushSizeChanged(UnityAction call)
```

Parameters

**call** UnityAction

## SubscribeToOnToolChanged(UnityAction)

Event invoked when selected tool changes.

```
public void SubscribeToOnToolChanged(UnityAction call)
```

Parameters

**call** UnityAction

# Class Tools

Namespace: [PAC.Drawing](#)

Defines how different tools act. I may rework this to be more like the BlendMode class where each Tool instance defines how it acts. Then to use a tool you would just do tool.Use(pixel) or similar.

```
public static class Tools
```

## Inheritance

[object](#) ← Tools

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### PencilLineSmoothing(Line, Line, bool)

Smooth the meeting point of the two lines (given as coords) so it is pixel-perfect - i.e. no hard 90-degree corner.

```
public static bool PencilLineSmoothing(Line line, Line previousLine,  
bool previousLineWasSmoothed)
```

#### Parameters

line [Line](#)

previousLine [Line](#)

previousLineWasSmoothed [bool](#)

#### Returns

[bool](#)

## UseBrush(File, int, int, IntVector2, IntVector2[], Color)

```
public static void UseBrush(File file, int layer, int frame, IntVector2 pixel,  
IntVector2[] brushBorderMaskPixels, Color colour)
```

### Parameters

file [File](#)

layer [int](#)

frame [int](#)

pixel [IntVector2](#)

brushBorderMaskPixels [IntVector2\[\]](#)

colour [Color](#)

## UseBrush(File, int, int, int, int, IntVector2[], Color)

```
public static void UseBrush(File file, int layer, int frame, int x, int y,  
IntVector2[] brushBorderMaskPixels, Color colour)
```

### Parameters

file [File](#)

layer [int](#)

frame [int](#)

x [int](#)

y [int](#)

brushBorderMaskPixels [IntVector2\[\]](#)

colour [Color](#)

## UseEyeDropper(File, int, int, IntVector2)

```
public static Color UseEyeDropper(File file, int layer, int frame, IntVector2 pixel)
```

Parameters

file [File](#)

layer [int](#)

frame [int](#)

pixel [IntVector2](#)

Returns

Color

## UseEyeDropper(File, int, int, int, int)

```
public static Color UseEyeDropper(File file, int layer, int frame, int x, int y)
```

Parameters

file [File](#)

layer [int](#)

frame [int](#)

x [int](#)

y [int](#)

Returns

Color

## UseFill(File, int, int, IntVector2, Color, int)

```
public static void UseFill(File file, int layer, int frame, IntVector2 pixel, Color colour, int maxNumOfIterations = 1000000)
```

## Parameters

file [File](#)

layer [int](#)

frame [int](#)

pixel [IntVector2](#)

colour [Color](#)

maxNumOfIterations [int](#)

## UseFill(File, int, int, int, int, Color, int)

```
public static void UseFill(File file, int layer, int frame, int x, int y, Color colour, int maxNumOfIterations = 1000000)
```

## Parameters

file [File](#)

layer [int](#)

frame [int](#)

x [int](#)

y [int](#)

colour [Color](#)

maxNumOfIterations [int](#)

## UsePattern(File, int, int, IPattern2D<Color32>)

```
public static void UsePattern(File file, int layer, int frame,  
IPattern2D<Color32> pattern)
```

## Parameters

file [File](#)

layer [int](#)

frame [int](#)

pattern [IPattern2D](#)<Color32>

## UsePattern(File, int, int, IPattern2D<Color>)

```
public static void UsePattern(File file, int layer, int frame,  
IPattern2D<Color> pattern)
```

## Parameters

file [File](#)

layer [int](#)

frame [int](#)

pattern [IPattern2D](#)<Color>

## UsePencil(File, int, int, IntVector2, Color)

```
public static void UsePencil(File file, int layer, int frame, IntVector2 pixel,  
Color colour)
```

## Parameters

file [File](#)

layer [int](#)

frame [int](#)

pixel [IntVector2](#)

colour Color

## UsePencil(File, int, int, int, int, Color)

```
public static void UsePencil(File file, int layer, int frame, int x, int y,  
Color colour)
```

### Parameters

file [File](#)

layer [int](#)

frame [int](#)

x [int](#)

y [int](#)

colour Color

## UseRubber(File, int, int, IntVector2)

```
public static void UseRubber(File file, int layer, int frame, IntVector2 pixel)
```

### Parameters

file [File](#)

layer [int](#)

frame [int](#)

pixel [IntVector2](#)

## UseRubber(File, int, int, IntVector2, IntVector2[])

```
public static void UseRubber(File file, int layer, int frame, IntVector2 pixel,  
IntVector2[] brushBorderMaskPixels)
```

### Parameters

file [File](#)

layer [int](#)

frame [int](#)

pixel [IntVector2](#)

brushBorderMaskPixels [IntVector2\[\]](#)

## UseRubber(File, int, int, int, int)

```
public static void UseRubber(File file, int layer, int frame, int x, int y)
```

### Parameters

file [File](#)

layer [int](#)

frame [int](#)

x [int](#)

y [int](#)

## UseRubber(File, int, int, int, int, IntVector2[])

```
public static void UseRubber(File file, int layer, int frame, int x, int y,  
IntVector2[] brushBorderMaskPixels)
```

## Parameters

file [File](#)

layer [int](#)

frame [int](#)

x [int](#)

y [int](#)

brushBorderMaskPixels [IntVector2\[\]](#)

## UseShape(File, int, int, IShape, Color)

```
public static void UseShape(File file, int layer, int frame, IShape shape,  
Color colour)
```

## Parameters

file [File](#)

layer [int](#)

frame [int](#)

shape [IShape](#)

colour [Color](#)

# Namespace PAC.EffectPanels

## Classes

### [BlurPanel](#)

A panel that blurs the view behind it.

### [EffectPanel](#)

A class to represent a type of panel that applies an effect to the view behind it - e.g. a blur panel or a pixellate panel.

### [PixellatePanel](#)

A panel that pixellates the view behind it.

# Class BlurPanel

Namespace: [PAC.EffectPanels](#)

A panel that blurs the view behind it.

```
public class BlurPanel : MonoBehaviour
```

## Inheritance

[object](#) ← BlurPanel

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### blurEnabled

```
public bool blurEnabled { get; }
```

Property Value

[bool](#)

## Methods

### EnableDisable(bool)

```
public void EnableDisable(bool enabled)
```

Parameters

enabled [bool](#)

# Class EffectPanel

Namespace: [PAC.EffectPanels](#)

A class to represent a type of panel that applies an effect to the view behind it - e.g. a blur panel or a pixellate panel.

```
public class EffectPanel : MonoBehaviour
```

## Inheritance

[object](#) ← EffectPanel

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Methods

### EnableDisable(bool)

```
public void EnableDisable(bool enabled)
```

#### Parameters

enabled [bool](#)

# Class PixellatePanel

Namespace: [PAC.EffectPanels](#)

A panel that pixellates the view behind it.

```
public class PixellatePanel : MonoBehaviour
```

## Inheritance

[object](#) ↗ ← PixellatePanel

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

pixellateOn

```
public bool pixellateOn { get; }
```

Property Value

[bool](#) ↗

# Namespace PAC.Exceptions

## Classes

### [TestException](#)

Indicates that an exception arose when running a test. Used to add an extra message, such as printing the test case, on top of the exception that occurred in the test.

### [UnderflowException](#)

### [UnreachableException](#)

Indicates that logically this code should not be reachable.

# Class TestException

Namespace: [PAC.Exceptions](#)

Indicates that an exception arose when running a test. Used to add an extra message, such as printing the test case, on top of the exception that occurred in the test.

```
public class TestException : AggregateException, ISerializable
```

## Inheritance

[object](#) ← [Exception](#) ← [AggregateException](#) ← [TestException](#)

## Implements

[ISerializable](#)

## Inherited Members

[AggregateException.Flatten\(\)](#) , [AggregateException.GetBaseException\(\)](#) ,  
[AggregateException.Handle\(Func<Exception, bool>\)](#) , [AggregateException.ToString\(\)](#) ,  
[AggregateException.InnerException](#) , [AggregateException.Message](#) ,  
[Exception.GetType\(\)](#) , [Exception.Data](#) , [Exception.HelpLink](#) , [Exception.HResult](#) ,  
[Exception.InnerException](#) , [Exception.Source](#) , [Exception.StackTrace](#) ,  
[Exception.TargetSite](#) , [Exception.SerializeObjectState](#) , [object.Equals\(object\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### TestException(string, Exception)

```
public TestException(string message, Exception innerException)
```

## Parameters

message [string](#)

innerException [Exception ↗](#)

# Class UnderflowException

Namespace: [PAC.Exceptions](#)

```
public class UnderflowException : ArithmeticException, ISerializable
```

## Inheritance

[object](#) ← [Exception](#) ← [SystemException](#) ← [ArithmeticException](#) ← [UnderflowException](#)

## Implements

[ISerializable](#)

## Inherited Members

[Exception.GetBaseException\(\)](#) , [Exception.GetType\(\)](#) , [Exception.ToString\(\)](#) ,  
[Exception.Data](#) , [Exception.HelpLink](#) , [Exception.HResult](#) , [Exception.InnerException](#) ,  
[Exception.Message](#) , [Exception.Source](#) , [Exception.StackTrace](#) , [Exception.TargetSite](#) ,  
[Exception.SerializeObjectState](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.GetHashCode\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### UnderflowException(string)

```
public UnderflowException(string message)
```

## Parameters

message [string](#)

# Class UnreachableException

Namespace: [PAC.Exceptions](#)

Indicates that logically this code should not be reachable.

```
public class UnreachableException : Exception, ISerializable
```

## Inheritance

[object](#) ← [Exception](#) ← UnreachableException

## Implements

[ISerializable](#)

## Inherited Members

[Exception.GetBaseException\(\)](#) , [Exception.GetType\(\)](#) , [Exception.ToString\(\)](#) ,  
[Exception.Data](#) , [Exception.HelpLink](#) , [Exception.HResult](#) , [Exception.InnerException](#) ,  
[Exception.Message](#) , [Exception.Source](#) , [Exception.StackTrace](#) , [Exception.TargetSite](#) ,  
[Exception.SerializeObjectState](#) , [object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) ,  
[object.GetHashCode\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### UnreachableException()

Indicates that logically this code should not be reachable.

```
public UnreachableException()
```

### UnreachableException(string)

Indicates that logically this code should not be reachable.

```
public UnreachableException(string message)
```

## Parameters

message [string](#)

# Namespace PAC.Extensions

## Classes

### [ColorExtensions](#)

Extension methods for Color.

### [ICollectionExtensions](#)

### [IDictionaryExtensions](#)

Extension methods for IDictionary<TKey, TValue>.

### [IEnumerableExtensions](#)

Extension methods for [IEnumerable<T>](#).

### [IPattern2DExtensions](#)

Extension methods for [IPattern2D<T>](#).

### [IReadOnlyContainsExtensions](#)

### [IReadOnlyListExtensions](#)

Extension methods for  [IReadOnlyList<T>](#).

### [ISetComparableExtensions](#)

Extension methods for [ISetComparable<T>](#).

### [IShapeExtensions](#)

Extension methods for [IShape](#).

### [IntRangeExtensions](#)

Extension methods involving [IntRange](#).

### [JsonConverters](#)

#### [JsonConverters.ColorJsonConverter](#)

Custom JSON converter for Color.

#### [JsonConverters.Texture2DJsonConverter](#)

Custom JSON converter for Texture2D.

#### [JsonConverters.Vector2JsonConverter](#)

Custom JSON converter for Vector2.

#### [JsonConverters.Vector3JsonConverter](#)

Custom JSON converter for Vector3.

## [KeyCodeExtensions](#)

Extension methods for Unity's KeyCode.

## [RandomExtensions](#)

Extension methods for [Random](#).

## [TextReaderExtensions](#)

## [Texture2DExtensions](#)

Extension methods for Unity's Texture2D.

## [TypeExtensions](#)

## [ValueTupleExtensions](#)

# Structs

## [Texture2DExtensions.ExtendCropOptions](#)

Options for [ExtendCrop\(Texture2D, in ExtendCropOptions\)](#).

# Class ColorExtensions

Namespace: [PAC.Extensions](#)

Extension methods for Color.

```
public static class ColorExtensions
```

## Inheritance

[object](#) ← ColorExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### Equals(Color, Color, float)

Returns whether each component of `colour` differs from the corresponding component of `otherColour` by <= `tolerance`.

```
public static bool Equals(this Color colour, Color otherColour, float tolerance)
```

#### Parameters

`colour` Color

`otherColour` Color

`tolerance` [float](#)

#### Returns

[bool](#)

# Class ICollectionExtensions

Namespace: [PAC.Extensions](#)

```
public static class ICollectionExtensions
```

## Inheritance

[object](#) ↴ ← ICollectionExtensions

## Inherited Members

[object.Equals\(object\)](#) ↴ , [object.Equals\(object, object\)](#) ↴ , [object.GetHashCode\(\)](#) ↴ ,  
[object.GetType\(\)](#) ↴ , [object.MemberwiseClone\(\)](#) ↴ , [object.ReferenceEquals\(object, object\)](#) ↴ ,  
[object.ToString\(\)](#) ↴

## Methods

### ContainsAll<T>(ICollection<T>, IEnumerable<T>)

Whether all the given elements are in the [ICollection<T>](#).

```
public static bool ContainsAll<T>(this ICollection<T> iCollection,  
IEnumerable<T> elements)
```

## Parameters

iCollection [ICollection](#) <T>

elements [IEnumerable](#) <T>

## Returns

[bool](#) ↴

## Type Parameters

T

## ContainsAll<T>(ICollection<T>, params T[])

Whether all the given elements are in the [ICollection<T>](#).

```
public static bool ContainsAll<T>(this ICollection<T> iCollection, params  
T[] elements)
```

Parameters

iCollection [ICollection](#)<T>

elements T[]

Returns

[bool](#)

Type Parameters

T

## ContainsAny<T>(ICollection<T>, IEnumerable<T>)

Whether any of the given elements are in the [ICollection<T>](#).

```
public static bool ContainsAny<T>(this ICollection<T> iCollection,  
IEnumerable<T> elements)
```

Parameters

iCollection [ICollection](#)<T>

elements [IEnumerable](#)<T>

Returns

[bool](#)

Type Parameters

T

## ContainsAny<T>(ICollection<T>, params T[])

Whether any of the given elements are in the [ICollection<T>](#).

```
public static bool ContainsAny<T>(this ICollection<T> iCollection, params  
T[] elements)
```

### Parameters

iCollection [ICollection](#)<T>

elements T[]

### Returns

[bool](#)

### Type Parameters

T

## ContainsNone<T>(ICollection<T>, IEnumerable<T>)

Whether none of the given elements are in the [ICollection<T>](#).

```
public static bool ContainsNone<T>(this ICollection<T> iCollection,  
IEnumerable<T> elements)
```

### Parameters

iCollection [ICollection](#)<T>

elements [IEnumerable](#)<T>

### Returns

[bool](#)

## Type Parameters

T

## ContainsNone<T>(ICollection<T>, params T[])

Whether none of the given elements are in the [ICollection<T>](#).

```
public static bool ContainsNone<T>(this ICollection<T> iCollection, params  
T[] elements)
```

## Parameters

iCollection [ICollection](#)<T>

elements T[]

## Returns

[bool](#)

## Type Parameters

T

## ContainsNotAll<T>(ICollection<T>, IEnumerable<T>)

Whether at least one of the given elements is not in the [ICollection<T>](#).

```
public static bool ContainsNotAll<T>(this ICollection<T> iCollection,  
IEnumerable<T> elements)
```

## Parameters

iCollection [ICollection](#)<T>

elements [IEnumerable](#)<T>

Returns

bool ↴

Type Parameters

T

**ContainsNotAll<T>(ICollection<T>, params T[])**

Whether at least one of the given elements is not in the [ICollection<T>](#).

```
public static bool ContainsNotAll<T>(this ICollection<T> iCollection, params  
T[] elements)
```

Parameters

iCollection [ICollection](#)<T>

elements T[]

Returns

bool ↴

Type Parameters

T

# Class IDictionaryExtensions

Namespace: [PAC.Extensions](#)

Extension methods for `IDictionary<TKey, TValue>`.

```
public static class IDictionaryExtensions
```

## Inheritance

[object](#) ← `IDictionaryExtensions`

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### `GetOrAdd<TKey, TValue>(IDictionary<TKey, TValue>, TKey, TValue)`

If the `IDictionary` already contains the key, it will return the existing value associated with it. If the key is not present, it will add it with the given value and return that value.

```
public static TValue GetOrAdd<TKey, TValue>(this IDictionary<TKey, TValue> dict,  
TKey key, TValue value)
```

#### Parameters

`dict` [IDictionary](#)<TKey, TValue>

`key` TKey

`value` TValue

#### Returns

TValue

## Type Parameters

TKey

TValue

# Class IEnumerableExtensions

Namespace: [PAC.Extensions](#)

Extension methods for [IEnumerable<T>](#).

```
public static class IEnumerableExtensions
```

## Inheritance

[object](#) ← IEnumerableExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### AreAllDistinct<T>(IEnumerable<T>)

Returns whether all the elements in the [IEnumerable<T>](#) are distinct. Uses the default equality comparer for type [T](#).

```
public static bool AreAllDistinct<T>(this IEnumerable<T> elements)
```

#### Parameters

[elements](#) [IEnumerable](#)<T>

#### Returns

[bool](#)

#### Type Parameters

[T](#)

## ArgMax<TElement, TCompare> (IEnumerable<TElement>, Func<TElement, TCompare>)

Applies the function to each element and returns the element that gives the highest output.  
If multiple elements give the highest output, the first one will be returned.

```
public static TElement ArgMax<TElement, TCompare>(this IEnumerable<TElement>
elements, Func<TElement, TCompare> function) where TCompare : IComparable<TCompare>
```

### Parameters

elements [IEnumerable](#)<TElement>

function [Func](#)<TElement, TCompare>

### Returns

TElement

### Type Parameters

TElement

TCompare

## ArgMin<TElement, TCompare> (IEnumerable<TElement>, Func<TElement, TCompare>)

Applies the function to each element and returns the element that gives the lowest output.  
If multiple elements give the lowest output, the first one will be returned.

```
public static TElement ArgMin<TElement, TCompare>(this IEnumerable<TElement>
elements, Func<TElement, TCompare> function) where TCompare : IComparable<TCompare>
```

### Parameters

elements [IEnumerable](#)<TElement>

```
function Func<TElement, TCompare>
```

Returns

TElement

Type Parameters

TElement

TCompare

**Concat<T>(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>[])**

Creates a new [IEnumerable<T>](#) that iterates through all of this one, then all of second, etc. Extends [Concat<TSource>\(IEnumerable<TSource>, IEnumerable<TSource>\)](#) to allow concating multiple [IEnumerable<T>](#)s.

```
public static IEnumerable<T> Concat<T>(this IEnumerable<T> first, IEnumerable<T> second, IEnumerable<T> third, params IEnumerable<T>[] subsequent)
```

Parameters

first [IEnumerable](#)<T>

second [IEnumerable](#)<T>

third [IEnumerable](#)<T>

subsequent [IEnumerable](#)<T>[]

Returns

[IEnumerable](#)<T>

Type Parameters

T

## ContainsAll<T>(IEnumerable<T>, IEnumerable<T>)

Whether all the given elements are in the [ICollection<T>](#).

```
public static bool ContainsAll<T>(this IEnumerable<T> iEnumerable,  
IEnumerable<T> elements)
```

### Parameters

iEnumerable [IEnumerable](#)<T>

elements [IEnumerable](#)<T>

### Returns

[bool](#)

### Type Parameters

T

## ContainsAll<T>(IEnumerable<T>, params T[])

Whether all the given elements are in the [IEnumerable<T>](#).

```
public static bool ContainsAll<T>(this IEnumerable<T> iEnumerable, params  
T[] elements)
```

### Parameters

iEnumerable [IEnumerable](#)<T>

elements T[]

### Returns

[bool](#)

### Type Parameters

T

## ContainsAny<T>(IEnumerable<T>, IEnumerable<T>)

Whether any of the given elements are in the [IEnumerable<T>](#).

```
public static bool ContainsAny<T>(this IEnumerable<T> iEnumerable,  
IEnumerable<T> elements)
```

### Parameters

iEnumerable [IEnumerable](#)<T>

elements [IEnumerable](#)<T>

### Returns

[bool](#)

### Type Parameters

T

## ContainsAny<T>(IEnumerable<T>, params T[])

Whether any of the given elements are in the [IEnumerable<T>](#).

```
public static bool ContainsAny<T>(this IEnumerable<T> iEnumerable, params  
T[] elements)
```

### Parameters

iEnumerable [IEnumerable](#)<T>

elements T[]

### Returns

[bool](#)

## Type Parameters

T

## ContainsNone<T>(IEnumerable<T>, IEnumerable<T>)

Whether none of the given elements are in the [IEnumerable<T>](#).

```
public static bool ContainsNone<T>(this IEnumerable<T> iEnumerable,  
IEnumerable<T> elements)
```

## Parameters

iEnumerable [IEnumerable](#)<T>

elements [IEnumerable](#)<T>

## Returns

[bool](#)

## Type Parameters

T

## ContainsNone<T>(IEnumerable<T>, params T[])

Whether none of the given elements are in the [IEnumerable<T>](#).

```
public static bool ContainsNone<T>(this IEnumerable<T> iEnumerable, params  
T[] elements)
```

## Parameters

iEnumerable [IEnumerable](#)<T>

elements T[]

Returns

[bool](#)

Type Parameters

T

## ContainsNotAll<T>(IEnumerable<T>, IEnumerable<T>)

Whether at least one of the given elements is not in the [ICollection<T>](#).

```
public static bool ContainsNotAll<T>(this IEnumerable<T> iEnumerable,  
IEnumerable<T> elements)
```

Parameters

iEnumerable [IEnumerable](#)<T>

elements [IEnumerable](#)<T>

Returns

[bool](#)

Type Parameters

T

## ContainsNotAll<T>(IEnumerable<T>, params T[])

Whether at least one of the given elements is not in the [IEnumerable<T>](#).

```
public static bool ContainsNotAll<T>(this IEnumerable<T> iEnumerable, params  
T[] elements)
```

Parameters

iEnumerable [IEnumerable](#)<T>

elements `T[]`

Returns

`bool`

Type Parameters

`T`

## CountIsAtLeast<T>(IEnumerable<T>, int)

Returns whether the given `IEnumerable<T>` has  $\geq n$  elements.

```
public static bool CountIsAtLeast<T>(this IEnumerable<T> elements, int n)
```

Parameters

elements `IEnumerable<T>`

n `int`

Returns

`bool`

Type Parameters

`T`

Remarks

Iterates through at most  $n$  elements of the `IEnumerable<T>`.

## CountIsAtMost<T>(IEnumerable<T>, int)

Returns whether the given `IEnumerable<T>` has  $\leq n$  elements.

```
public static bool CountIsAtMost<T>(this IEnumerable<T> elements, int n)
```

## Parameters

elements [IEnumerable<T>](#)

n [int](#)

## Returns

[bool](#)

## Type Parameters

T

## Remarks

Iterates through at most n + 1 elements of the [IEnumerable<T>](#).

## CountIsExactly<T>(IEnumerable<T>, int)

Returns whether the given [IEnumerable<T>](#) has exactly n elements.

```
public static bool CountIsExactly<T>(this IEnumerable<T> elements, int n)
```

## Parameters

elements [IEnumerable<T>](#)

n [int](#)

## Returns

[bool](#)

## Type Parameters

T

## Remarks

Iterates through at most n + 1 elements of the [IEnumerable<T>](#).

# EnumerateOccurrences<T>(IEnumerable<T>)

Pairs each element with the number of times that element has already occurred.

```
char[] actual = new char[] { 'a', 'b', 'a', 'c', 'b', 'a' }.EnumerateOccurrences();
char[] expected = new (char, int)[] { ('a', 0), ('b', 0), ('a', 1), ('c', 0), ('b', 1), ('a', 2) };
expected.SequenceEqual(actual); // true

public static IEnumerable<T element, int occurrence> EnumerateOccurrences<T>(this
IEnumerable<T> elements)
```

Parameters

`elements` [IEnumerable<T>](#)

Returns

[IEnumerable<T element, int index>](#)

Type Parameters

`T`

Remarks

Uses the default equality comparer for type `T`.

# Enumerate<T>(IEnumerable<T>)

Returns `(elements[0], 0), (elements[1], 1), ..., (elements[^1], elements.Count - 1)`.

```
public static IEnumerable<T element, int index> Enumerate<T>(this
IEnumerable<T> elements)
```

Parameters

`elements` [IEnumerable<T>](#)

Returns

[IEnumerable<\(T element, int index\)>](#)

Type Parameters

T

## Flatten<T>(IEnumerable<IEnumerable<T>>)

Flattens a sequence of sequences into one sequence, by first iterating over the first sequence, then the second sequence, etc. For example, { { 1, 2, 3 }, { 4, 5 }, { }, { 6, 7, 8, 9, 10 } } flattens to { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }.

```
public static IEnumerable<T> Flatten<T>(this IEnumerable<IEnumerable<T>> sequences)
```

Parameters

sequences [IEnumerable<IEnumerable<T>>](#)

Returns

[IEnumerable<T>](#)

Type Parameters

T

Exceptions

[ArgumentNullException](#)

sequences is null.

[ArgumentException](#)

One of the sequences in sequences is null.

## IsSubsequenceOf<T>(IEnumerable<T>, IEnumerable<T>)

Returns whether this [IEnumerable<T>](#) is a subsequence of [otherSequence](#); that is, [otherSequence](#) is this [IEnumerable<T>](#) with potentially extra elements added anywhere in the sequence.

```
public static bool IsSubsequenceOf<T>(this IEnumerable<T> sequence,  
IEnumerable<T> otherSequence)
```

### Parameters

sequence [IEnumerable<T>](#)

otherSequence [IEnumerable<T>](#)

### Returns

[bool](#)

### Type Parameters

T

### See Also

[IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#).

## IsSupersequenceOf<T>(IEnumerable<T>, IEnumerable<T>)

Returns whether this [IEnumerable<T>](#) is a supersequence of [otherSequence](#); that is, this [IEnumerable<T>](#) is [otherSequence](#) with potentially extra elements added anywhere in the sequence.

```
public static bool IsSupersequenceOf<T>(this IEnumerable<T> sequence,  
IEnumerable<T> otherSequence)
```

### Parameters

sequence [IEnumerable](#)<T>

otherSequence [IEnumerable](#)<T>

Returns

[bool](#)

Type Parameters

T

### See Also

[IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#).

## MinAndMax<T>(IEnumerable<T>)

Returns the minimum and maximum element of the [IEnumerable](#)<T>. Can be more efficient than calling [Min<TSource>\(IEnumerable<TSource>\)](#) and [Max<TSource>\(IEnumerable<TSource>\)](#) as this only iterates through the [IEnumerable](#)<T> once.

Throws an exception if T doesn't have a default comparer.

```
public static (T min, T max) MinAndMax<T>(this IEnumerable<T> elements)
```

Parameters

elements [IEnumerable](#)<T>

Returns

(T, T)

Type Parameters

T

## None<T>(IEnumerable<T>)

Returns true iff the given sequence is empty.

```
public static bool None<T>(this IEnumerable<T> elements)
```

Parameters

elements [IEnumerable](#)<T>

Returns

[bool](#)

Type Parameters

T

## None<T>(IEnumerable<T>, Func<T, bool>)

Returns true iff none of the elements satisfy the predicate.

```
public static bool None<T>(this IEnumerable<T> elements, Func<T, bool> predicate)
```

Parameters

elements [IEnumerable](#)<T>

predicate [Func](#)<T, [bool](#)>

Returns

[bool](#)

Type Parameters

T

## NotAll<T>(IEnumerable<T>, Func<T, bool>)

Returns true iff at least one of the elements doesn't satisfy the predicate.

```
public static bool NotAll<T>(this IEnumerable<T> elements, Func<T, bool> predicate)
```

Parameters

elements [IEnumerable](#)<T>

predicate [Func](#)<T, [bool](#)>

Returns

[bool](#)

Type Parameters

T

## Product(IEnumerable<decimal>)

Returns the product of all elements in the sequence, or 1 if the sequence is empty.

```
public static decimal Product(this IEnumerable<decimal> elements)
```

Parameters

elements [IEnumerable](#)<[decimal](#)>

Returns

[decimal](#)

## Product(IEnumerable<double>)

Returns the product of all elements in the sequence, or 1 if the sequence is empty.

```
public static double Product(this IEnumerable<double> elements)
```

Parameters

`elements` [IEnumerable](#)<[double](#)>

Returns

[double](#)

## Product(IEnumerable<int>)

Returns the product of all elements in the sequence, or 1 if the sequence is empty.

```
public static int Product(this IEnumerable<int> elements)
```

Parameters

`elements` [IEnumerable](#)<[int](#)>

Returns

[int](#)

## Product(IEnumerable<long>)

Returns the product of all elements in the sequence, or 1 if the sequence is empty.

```
public static long Product(this IEnumerable<long> elements)
```

Parameters

`elements` [IEnumerable](#)<[long](#)>

Returns

[long](#)

## Product(IEnumerable<float>)

Returns the product of all elements in the sequence, or 1 if the sequence is empty.

```
public static float Product(this IEnumerable<float> elements)
```

Parameters

elements [IEnumerable<T>](#)

Returns

[float](#)

## RepeatIEnumerable<T>(IEnumerable<T>)

Creates an [IEnumerable<T>](#) that repeats the given [IEnumerable<T>](#) indefinitely, looping back round to the beginning when it reaches the end.

```
public static IEnumerable<T> RepeatIEnumerable<T>(IEnumerable<T> elements)
```

Parameters

elements [IEnumerable<T>](#)

Returns

[IEnumerable<T>](#)

Type Parameters

T

## Repeat<T>(T)

Creates an [IEnumerable<T>](#) that repeats the given element indefinitely.

```
public static IEnumerable<T> Repeat<T>(T element)
```

Parameters

element T

Returns

[IEnumerable](#)<T>

Type Parameters

T

## Replace<T>(IEnumerable<T>, T, T)

Replaces all occurrences of `toReplace` with `replaceWith`. Uses the default equality comparer for type `T`.

```
public static IEnumerable<T> Replace<T>(this IEnumerable<T> elements, T toReplace,  
T replaceWith)
```

Parameters

elements [IEnumerable](#)<T>

toReplace T

replaceWith T

Returns

[IEnumerable](#)<T>

Type Parameters

T

## Singleton<T>(T)

Creates an [IEnumerable](#)<T> with only one element.

```
public static IEnumerable<T> Singleton<T>(T element)
```

Parameters

element T

Returns

[IEnumerable](#)<T>

Type Parameters

T

## SkipIndex<T>(IEnumerable<T>, int)

Returns the given sequence but without the element at the given index. If the index is outside the range of the sequence, the sequence will be unaffected.

```
public static IEnumerable<T> SkipIndex<T>(this IEnumerable<T> elements, int index)
```

Parameters

elements [IEnumerable](#)<T>

index int

Returns

[IEnumerable](#)<T>

Type Parameters

T

## ToPrettyString<T>(IEnumerable<T>)

Formats `elements` into a string of the form "`{ elements[0], ..., elements[^1] }`" (with the '...' replaced by `elements`).

```
public static string ToPrettyString<T>(this IEnumerable<T> elements)
```

Parameters

`elements` [IEnumerable](#)<T>

Returns

[string](#)

Type Parameters

T

## ZipCurrentAndNext<T>(IEnumerable<T>)

Returns `(elements[0], elements[1]), (elements[1], elements[2]), ..., (elements[^2], elements[^1])`.

Returns an empty [IEnumerable](#)<T> if `elements` has length 0 or 1.

```
public static IEnumerable<(T current, T next)> ZipCurrentAndNext<T>(this
IEnumerable<T> elements)
```

Parameters

`elements` [IEnumerable](#)<T>

Returns

[IEnumerable](#)<(T [first](#), T [second](#))>

Type Parameters

T

# Zip<T1, T2>(IEnumerable<T1>, IEnumerable<T2>)

Returns `(left[0], right[0]), (left[1], right[1]), ...`, until either `left` or `right` ends.

```
public static IEnumerable<(T1 left, T2 right)> Zip<T1, T2>(this IEnumerable<T1>
    left, IEnumerable<T2> right)
```

## Parameters

`left` [IEnumerable](#)<T1>

`right` [IEnumerable](#)<T2>

## Returns

[IEnumerable](#)<(T1 `left`, T2 `right`)>

## Type Parameters

T1

T2

# Class IPattern2DExtensions

Namespace: [PAC.Extensions](#)

Extension methods for [IPattern2D<T>](#).

```
public static class IPattern2DExtensions
```

## Inheritance

[object](#) ← IPattern2DExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### ToTexture(IPattern2D<Color32>, IntRect)

Turns the section of the pattern in the given rect into a Unity Texture2D.

```
public static Texture2D ToTexture(this IPattern2D<Color32> pattern,  
IntRect textureRect)
```

#### Parameters

pattern [IPattern2D<Color32>](#)

textureRect [IntRect](#)

#### Returns

Texture2D

#### Remarks

Uses Texture2D.SetPixels32(Color32[]), so should be faster than  
[ToTexture\(IPattern2D<Color>, IntRect\)](#) which uses Texture2D.SetPixels(Color[]).

# ToTexture(IPattern2D<Color>, IntRect)

Turns the section of the pattern in the given rect into a Unity Texture2D.

```
public static Texture2D ToTexture(this IPattern2D<Color> pattern,  
IntRect textureRect)
```

## Parameters

**pattern** [IPattern2D<Color>](#)

**textureRect** [IntRect](#)

## Returns

Texture2D

## Remarks

Uses Texture2D.SetPixels(Color[]), so won't be as fast as [ToTexture\(IPattern2D<Color32>, IntRect\)](#) which uses Texture2D.SetPixels32(Color32[]).

# Class IReadOnlyContainsExtensions

Namespace: [PAC.Extensions](#)

```
public static class IReadOnlyContainsExtensions
```

## Inheritance

[object](#) ← IReadOnlyContainsExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### ContainsAll<T>(IReadOnlyContains<T>, IEnumerable<T>)

Whether all the given elements are in the [IReadOnlyContains<T>](#).

```
public static bool ContainsAll<T>(this IReadOnlyContains<T> iReadOnlyContains,  
IEnumerable<T> elements)
```

## Parameters

iReadOnlyContains [IReadOnlyContains<T>](#)

elements [IEnumerable<T>](#)

## Returns

[bool](#)

## Type Parameters

T

## ContainsAll<T>(IReadOnlyContains<T>, params T[])

Whether all the given elements are in the [IReadOnlyContains<T>](#).

```
public static bool ContainsAll<T>(this IReadOnlyContains<T> iReadOnlyContains,  
params T[] elements)
```

Parameters

iReadOnlyContains [IReadOnlyContains<T>](#)

elements T[]

Returns

[bool](#)

Type Parameters

T

## ContainsAny<T>(IReadOnlyContains<T>, IEnumerable<T>)

Whether any of the given elements are in the [IReadOnlyContains<T>](#).

```
public static bool ContainsAny<T>(this IReadOnlyContains<T> iReadOnlyContains,  
IEnumerable<T> elements)
```

Parameters

iReadOnlyContains [IReadOnlyContains<T>](#)

elements [IEnumerable](#)<T>

Returns

[bool](#)

## Type Parameters

T

## ContainsAny<T>(IReadOnlyContains<T>, params T[])

Whether any of the given elements are in the [IReadOnlyContains<T>](#).

```
public static bool ContainsAny<T>(this IReadOnlyContains<T> iReadOnlyContains,  
params T[] elements)
```

## Parameters

iReadOnlyContains [IReadOnlyContains<T>](#)

elements T[]

## Returns

[bool](#)

## Type Parameters

T

## ContainsNone<T>(IReadOnlyContains<T>, IEnumerable<T>)

Whether none of the given elements are in the [IReadOnlyContains<T>](#).

```
public static bool ContainsNone<T>(this IReadOnlyContains<T> iReadOnlyContains,  
IEnumerable<T> elements)
```

## Parameters

iReadOnlyContains [IReadOnlyContains<T>](#)

elements [IEnumerable<T>](#)

Returns

[bool](#)

Type Parameters

T

## ContainsNone<T>(IReadOnlyContains<T>, params T[])

Whether none of the given elements are in the [IReadOnlyContains<T>](#).

```
public static bool ContainsNone<T>(this IReadOnlyContains<T> iReadOnlyContains,  
params T[] elements)
```

Parameters

iReadOnlyContains [IReadOnlyContains<T>](#)

elements T[]

Returns

[bool](#)

Type Parameters

T

## ContainsNotAll<T>(IReadOnlyContains<T>, IEnumerable<T>)

Whether at least one of the given elements is not in the [IReadOnlyContains<T>](#).

```
public static bool ContainsNotAll<T>(this IReadOnlyContains<T> iReadOnlyContains,  
IEnumerable<T> elements)
```

Parameters

`iReadOnlyContains` [IReadOnlyContains<T>](#)

`elements` [IEnumerable<T>](#)

Returns

[bool](#)

Type Parameters

`T`

## ContainsNotAll<T>(IReadOnlyContains<T>, params T[])

Whether at least one of the given elements is not in the [IReadOnlyContains<T>](#).

```
public static bool ContainsNotAll<T>(this IReadOnlyContains<T> iReadOnlyContains,  
params T[] elements)
```

Parameters

`iReadOnlyContains` [IReadOnlyContains<T>](#)

`elements` `T[]`

Returns

[bool](#)

Type Parameters

`T`

# Class IReadOnlyListExtensions

Namespace: [PAC.Extensions](#)

Extension methods for [IReadOnlyList<T>](#).

```
public static class IReadOnlyListExtensions
```

## Inheritance

[object](#) ← IReadOnlyListExtensions

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#),  
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#),  
[object.ToString\(\)](#)

## Methods

### GetRange<T>(IReadOnlyList<T>, int, int)

Lazily iterates over the [IReadOnlyList<T>](#)'s elements from position [index](#) (inclusive) to [index + count](#) (exclusive). This is defined even if [count](#) is negative.

```
public static IEnumerable<T> GetRange<T>(this IReadOnlyList<T> list, int index,  
int count)
```

#### Parameters

[list](#) [IReadOnlyList](#)<T>

[index](#) [int](#)

The index to start iterating from.

[count](#) [int](#)

The number of elements to iterate over, starting from [index](#). If negative, it will iterate from [index](#) backwards.

Returns

[IEnumerable](#)<T>

Type Parameters

T

Exceptions

[ArgumentNullException](#)

list is empty.

## GetRange<T>(IReadOnlyList<T>, Range)

Lazily iterates over the indices in the [Range](#) and yields the [IReadOnlyList](#)'s element at that position.

```
public static IEnumerable<T> GetRange<T>(this IReadOnlyList<T> list, Range indices)
```

Parameters

list [IReadOnlyList](#)<T>

indices [Range](#)

The range of indices to iterate over.

Returns

[IEnumerable](#)<T>

Type Parameters

T

Exceptions

[ArgumentNullException](#)

`list` is empty.

# Class ISetComparableExtensions

Namespace: [PAC.Extensions](#)

Extension methods for [ISetComparable<T>](#).

```
public static class ISetComparableExtensions
```

## Inheritance

[object](#) ← ISetComparableExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### IsProperSubsetOf<T1, T2>(T1, T2)

Whether **set** as a set is a proper subset of **other** as a set.

```
public static bool IsProperSubsetOf<T1, T2>(this T1 set, T2 other) where T1 : ISetComparable<T2>
```

#### Parameters

**set** T1

**other** T2

#### Returns

[bool](#)

#### Type Parameters

T1

## Remarks

Recall that '`set` is proper subset of `other`' means that `set` is a subset of `other`, but is not set-equal to `other`. In more detail, every element of `set` is an element of `other`, ignoring order and duplicate elements, but there is at least one element of `other` that is not in `set`.

## See Also

[IsProperSupersetOf<T1, T2>\(T1, T2\)](#), [IsSubsetOf\(T\)](#), [SetEquals\(T\)](#)

## IsProperSupersetOf<T1, T2>(T1, T2)

Whether `set` as a set is a proper superset of `other` as a set.

```
public static bool IsProperSupersetOf<T1, T2>(this T1 set, T2 other) where T1 : ISetComparable<T2>
```

## Parameters

`set` T1

`other` T2

## Returns

[bool](#)

## Type Parameters

T1

T2

## Remarks

Recall that '`set` is proper superset of `other`' means that `set` is a superset of `other`, but is not set-equal to `other`. In more detail, every element of `other` is an element of `set`, ignoring order and duplicate elements, but there is at least one element of `set` that is not in `other`.

## See Also

[IsProperSupersetOf<T1, T2>\(T1, T2\)](#), [IsSubsetOf\(T\)](#), [SetEquals\(T\)](#)

# Class IShapeExtensions

Namespace: [PAC.Extensions](#)

Extension methods for [IShape](#).

```
public static class IShapeExtensions
```

## Inheritance

[object](#) ← IShapeExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### ToTexture(IShape, Color)

Turns the pixels in the shape's bounding rect into a Unity Texture2D with the given colour.

```
public static Texture2D ToTexture(this IShape shape, Color colour)
```

#### Parameters

shape [IShape](#)

colour Color

#### Returns

Texture2D

### ToTexture(IShape, Color, IntRect)

Turns the pixels in the given rect into a Unity Texture2D with the given colour, using any of the shape's points that lie within that rect.

```
public static Texture2D ToTexture(this IShape shape, Color colour,  
IntRect textureRect)
```

## Parameters

shape [IShape](#)

colour Color

textureRect [IntRect](#)

## Returns

Texture2D

# Class IntRangeExtensions

Namespace: [PAC.Extensions](#)

Extension methods involving [IntRange](#).

```
public static class IntRangeExtensions
```

## Inheritance

[object](#) ← IntRangeExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### GetRange<T>(IReadOnlyList<T>, IntRange)

Lazily iterates over the indices in the [IntRange](#) and yields the [IReadOnlyList<T>](#)'s element at that position.

```
public static IEnumerable<T> GetRange<T>(this IReadOnlyList<T> list,  
IntRange indices)
```

#### Parameters

list [IReadOnlyList](#)<T>

indices [IntRange](#)

The range of indices to iterate over.

#### Returns

[IEnumerable](#)<T>

#### Type Parameters

## Exceptions

### [ArgumentNullException](#)

`list` is empty.

## Next(Random, IntRange)

Returns a random integer within the given range.

```
public static int Next(this Random random, IntRange range)
```

### Parameters

`random` [Random](#)

`range` [IntRange](#)

### Returns

[int](#)

## Random(Random, IntRange)

Creates an infinite random sequence.

```
public static IEnumerable<int> Random(Random random, IntRange range)
```

### Parameters

`random` [Random](#)

`range` [IntRange](#)

### Returns

[IEnumerable](#)<[int](#)>



# Class JsonConverters

Namespace: [PAC.Extensions](#)

```
public static class JsonConverters
```

## Inheritance

[object](#) ← JsonConverters

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Fields

### allConverters

All the custom JSON converters define in this class.

```
public static readonly JsonConversion.JsonConverterSet allConverters
```

## Field Value

[JsonConversion.JsonConverterSet](#)

# Class JsonConverters.ColorJsonConverter

Namespace: [PAC.Extensions](#)

Custom JSON converter for Color.

```
public class JsonConverters.ColorJsonConverter : JsonConversion.JsonConverter<Color,  
JsonData.List>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter](#)<Color, [JsonData.List](#)> ←  
JsonConverters.ColorJsonConverter

## Inherited Members

[JsonConversion.JsonConverter<Color, jsonData.List>.ToJson\(Color\)](#) ,  
[JsonConversion.JsonConverter<Color, jsonData.List>.FromJson\(jsonData.List\)](#) ,  
[JsonConversion.JsonConverter<Color, jsonData.List>.FromJson\(jsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### FromJson(List)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override Color FromJson(JsonData.List jsonData)
```

## Parameters

jsonData [JsonData.List](#)

Returns

Color

## ToJson(Color)

Attempts to convert the C# object into JSON data.

```
public override JsonData.List ToJson(Color color)
```

Parameters

color Color

Returns

[JsonData.List](#)

# Class JsonConverters.Texture2DJsonConverter

Namespace: [PAC.Extensions](#)

Custom JSON converter for Texture2D.

```
public class JsonConverters.Texture2DJsonConverter :  
JsonConversion.JsonConverter<Texture2D, JsonData.Object>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter](#)<Texture2D, [JsonData.Object](#)> ←  
JsonConverters.Texture2DJsonConverter

## Inherited Members

[JsonConversion.JsonConverter<Texture2D, JsonData.Object>.ToJson\(Texture2D\)](#) ,  
[JsonConversion.JsonConverter<Texture2D, JsonData.Object>.FromJson\(JsonData.Object\)](#) ,  
[JsonConversion.JsonConverter<Texture2D, JsonData.Object>.FromJson\(JsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### FromJson(Object)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override Texture2D FromJson(JsonData.Object jsonData)
```

Parameters

`jsonData` [JsonData.Object](#)

Returns

Texture2D

## ToJson(Texture2D)

Attempts to convert the C# object into JSON data.

```
public override JsonData.Object ToJson(Texture2D tex)
```

Parameters

`tex` Texture2D

Returns

[JsonData.Object](#)

# Class JsonConverters.Vector2JsonConverter

Namespace: [PAC.Extensions](#)

Custom JSON converter for Vector2.

```
public class JsonConverters.Vector2JsonConverter :  
JsonConversion.JsonConverter<Vector2, jsonData.List>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter](#)<Vector2, jsonData.List> ←  
JsonConverters.Vector2JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<Vector2, jsonData.List>.ToJson\(Vector2\)](#) ,  
[JsonConversion.JsonConverter<Vector2, jsonData.List>.FromJson\(jsonData.List\)](#) ,  
[JsonConversion.JsonConverter<Vector2, jsonData.List>.FromJson\(jsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### FromJson(List)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override Vector2 FromJson(jsonData.List jsonData)
```

Parameters

**jsonData** [JsonData.List](#)

Returns

Vector2

## ToJson(Vector2)

Attempts to convert the C# object into JSON data.

```
public override JsonData.List ToJson(Vector2 vec)
```

Parameters

**vec** Vector2

Returns

[JsonData.List](#)

# Class JsonConverters.Vector3JsonConverter

Namespace: [PAC.Extensions](#)

Custom JSON converter for Vector3.

```
public class JsonConverters.Vector3JsonConverter :  
JsonConversion.JsonConverter<Vector3, jsonData.List>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter](#)<Vector3, jsonData.List> ←  
JsonConverters.Vector3JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<Vector3, jsonData.List>.ToJson\(Vector3\)](#) ,  
[JsonConversion.JsonConverter<Vector3, jsonData.List>.FromJson\(jsonData.List\)](#) ,  
[JsonConversion.JsonConverter<Vector3, jsonData.List>.FromJson\(jsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### FromJson(List)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override Vector3 FromJson(jsonData.List jsonData)
```

Parameters

**jsonData** [JsonData.List](#)

Returns

Vector3

## ToJson(Vector3)

Attempts to convert the C# object into JSON data.

```
public override JsonData.List ToJson(Vector3 vec)
```

Parameters

**vec** Vector3

Returns

[JsonData.List](#)

# Class KeyCodeExtensions

Namespace: [PAC.Extensions](#)

Extension methods for Unity's KeyCode.

```
public static class KeyCodeExtensions
```

## Inheritance

[object](#) ← KeyCodeExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### Parse(string)

Parses the given [string](#) as a Unity KeyCode.

```
public static KeyCode Parse(string str)
```

Parameters

**str** [string](#)

Returns

KeyCode

Exceptions

[ArgumentException](#)

**str** couldn't been parsed as a KeyCode.

# TryParse(string, out KeyCode)

Tries parsing the given [string](#) as a Unity KeyCode, returning whether the parsing was successful.

```
public static bool TryParse(string str, out KeyCode parsed)
```

## Parameters

**str** [string](#)

**parsed** KeyCode

The result of the parsing, if successful. If unsuccessful, it will be KeyCode.None.

## Returns

[bool](#)

Whether the parsing was successful.

# Class RandomExtensions

Namespace: [PAC.Extensions](#)

Extension methods for [Random](#).

```
public static class RandomExtensions
```

## Inheritance

[object](#) ← RandomExtensions

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#),  
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#),  
[object.ToString\(\)](#)

## Methods

### NextBool(Random)

Generates a uniformly random [bool](#) value.

```
public static bool NextBool(this Random random)
```

#### Parameters

random [Random](#)

#### Returns

[bool](#)

### NextElement<T>(Random, IReadOnlyList<T>)

Selects a uniformly random element from [elements](#).

```
public static T NextElement<T>(this Random random, IReadOnlyList<T> elements)
```

## Parameters

random [Random](#)

elements [IReadOnlyList](#)<T>

## Returns

T

## Type Parameters

T

## Exceptions

[ArgumentException](#)

elements is empty.

## ToSequence(Random)

Creates an infinite random sequence, lazily generated using [Next\(\)](#).

```
public static IEnumerable<int> ToSequence(this Random random)
```

## Parameters

random [Random](#)

## Returns

[IEnumerable](#)<int>

## Remarks

This does not deep-copy the [Random](#), so calling [Next\(\)](#) on this [Random](#) between the generation of two elements will affect the sequence. This also means that iterating over the

sequence more than once will most likely give a different sequence each time; you may want to 'save' a portion of the sequence using `new Random().ToSequence().Take(10).ToArray()` or similar.

## ToSequence(Random, int, int)

Creates an infinite random sequence, lazily generated using [Next\(int, int\)](#).

```
public static IEnumerable<int> ToSequence(this Random random, int minValue, int maxValue)
```

### Parameters

`random` [Random](#)

`minValue` [int](#)

The inclusive lower bound of the random numbers generated.

`maxValue` [int](#)

The exclusive upper bound of the random numbers generated. `maxValue` must be greater than or equal to `minValue`.

### Returns

[IEnumerable](#)<[int](#)>

### Remarks

This does not deep-copy the [Random](#), so calling [Next\(int, int\)](#) on this [Random](#) between the generation of two elements will affect the sequence. This also means that iterating over the sequence more than once will most likely give a different sequence each time; you may want to 'save' a portion of the sequence using `new Random().ToSequence(minValue, maxValue).Take(10).ToArray()` or similar.

### Exceptions

[ArgumentOutOfRangeException](#)

`minValue` is greater than `maxValue`.

# Class TextReaderExtensions

Namespace: [PAC.Extensions](#)

```
public static class TextReaderExtensions
```

## Inheritance

[object](#) ← TextReaderExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### ReadMatch(TextReader, IEnumerable<char>)

Reads, checking if it reads the given sequence of characters (starting at the current point). Returns the characters it matched before it found a non-match. Doesn't reach the non-match character / the next character after all have characters been matched.

```
public static string ReadMatch(this TextReader reader, IEnumerable<char> text)
```

#### Parameters

reader [TextReader](#)

text [IEnumerable](#)<[char](#)>

#### Returns

[string](#)

#### Exceptions

[EndOfStreamException](#)

If there is nothing left for the reader to read at the start of the method.

## ReadMatch(TextReader, IEnumerable<ICollection<char>>)

Reads, checking if the first read character is in the first collection of characters, then if the second is in the second collection, etc. until all character collections have been matched or until a character doesn't match. Returns the characters it matched before it found a non-match. Doesn't reach the non-match character / the next character after all have characters been matched.

```
public static string ReadMatch(this TextReader reader,  
IEnumerable<ICollection<char>> charSets)
```

### Parameters

reader [TextReader](#)

charSets [IEnumerable](#)<[ICollection](#)<[char](#)>>

### Returns

[string](#)

### Exceptions

[EndOfStreamException](#)

If there is nothing left for the reader to read at the start of the method.

## ReadMatch(TextReader, IEnumerable<Func<char, bool>>)

Reads, checking if the first read character satisfies the first match condition, then if the second matches the second condition, etc. until all character collections have been matched or until a character doesn't match. Returns the characters it matched before it found a non-match. Doesn't reach the non-match character / the next character after all have characters been matched.

```
public static string ReadMatch(this TextReader reader, IEnumerable<Func<char, bool>> matchConditions)
```

## Parameters

reader [TextReader](#)

matchConditions [IEnumerable<Func<char, bool>>](#)

## Returns

[string](#)

## Exceptions

[EndOfStreamException](#)

If there is nothing left for the reader to read at the start of the method.

## ReadMatch(TextReader, string)

Reads, checking if it reads the given string (starting at the current point). Returns the characters it matched before it found a non-match. Doesn't reach the non-match character / the next character after all have characters been matched.

```
public static string ReadMatch(this TextReader reader, string text)
```

## Parameters

reader [TextReader](#)

text [string](#)

## Returns

[string](#)

## Exceptions

[EndOfStreamException](#)

If there is nothing left for the reader to read at the start of the method.

## ReadMatchAll(TextReader, IEnumerable<char>)

Reads, checking if it reads the given sequence of characters (starting at the current point).

```
public static bool ReadMatchAll(this TextReader reader, IEnumerable<char> text)
```

Parameters

reader [TextReader](#)

text [IEnumerable](#)<[char](#)>

Returns

[bool](#)

Exceptions

[EndOfStreamException](#)

If there is nothing left for the reader to read at the start of the method.

## ReadMatchAll(TextReader, IEnumerable<ICollection<char>>)

Reads, checking if the first read character is in the first collection of characters, then if the second is in the second collection, etc. until all character collections have been matched. Doesn't reach the non-match character / the next character after all have characters been matched.

```
public static bool ReadMatchAll(this TextReader reader,  
IEnumerable<ICollection<char>> charSets)
```

Parameters

reader [TextReader](#)

charSets [IEnumerable](#)<[ICollection](#)<char>>>

Returns

[bool](#)

Exceptions

[EndOfStreamException](#)

If there is nothing left for the reader to read at the start of the method.

## ReadMatchAll(TextReader, string)

Reads, checking if it reads the given string (starting at the current point). Doesn't reach the non-match character / the next character after all have characters been matched. Doesn't reach the non-match character / the next character after all have characters been matched.

```
public static bool ReadMatchAll(this TextReader reader, string text)
```

Parameters

reader [TextReader](#)

text [string](#)

Returns

[bool](#)

Exceptions

[EndOfStreamException](#)

If there is nothing left for the reader to read at the start of the method.

## TryReadIntoChar(TextReader, ref char)

If the TextReader can read another character, it will read it into chr and return true. Otherwise, it will return false without changing chr and without reading.

```
public static bool TryReadIntoChar(this TextReader reader, ref char chr)
```

Parameters

reader [TextReader](#)

chr [char](#)

Returns

[bool](#)

# Class Texture2DExtensions

Namespace: [PAC.Extensions](#)

Extension methods for Unity's Texture2D.

```
public static class Texture2DExtensions
```

## Inheritance

[object](#) ← Texture2DExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### Applied(Texture2D)

Calls Texture2D.Apply() on the Texture2D then returns it.

```
public static Texture2D Applied(this Texture2D texture)
```

#### Parameters

**texture** Texture2D

#### Returns

Texture2D

### Blend(Color, Texture2D, BlendMode)

Blends **topColour** onto each pixel of a deep copy of **bottomTexture** using the given [Blend Mode](#).

```
public static Texture2D Blend(Color topColour, Texture2D bottomTexture,  
BlendMode blendMode)
```

## Parameters

**topColour** Color

**bottomTexture** Texture2D

**blendMode** [BlendMode](#)

## Returns

Texture2D

## Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## Blend(Texture2D, Texture2D, BlendMode)

Blends a deep copy of **topTexture** onto a deep copy of **bottomTexture** using the given [Blend Mode](#), placing the bottom-left corner of **topTexture** on the bottom-left corner of **bottomTexture**.

```
public static Texture2D Blend(this Texture2D topTexture, Texture2D bottomTexture,  
BlendMode blendMode)
```

## Parameters

**topTexture** Texture2D

**bottomTexture** Texture2D

**blendMode** [BlendMode](#)

## Returns

Texture2D

## Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## Blend(Texture2D, Texture2D, BlendMode, IntVector2)

Blends a deep copy of `topTexture` onto a deep copy of `bottomTexture` using the given [Blend Mode](#), placing the bottom-left corner of `topTexture` at the coordinates `topTextureOffset` on `bottomTexture`.

```
public static Texture2D Blend(this Texture2D topTexture, Texture2D bottomTexture,  
BlendMode blendMode, IntVector2 topTextureOffset)
```

## Parameters

`topTexture` Texture2D

`bottomTexture` Texture2D

`blendMode` [BlendMode](#)

`topTextureOffset` [IntVector2](#)

## Returns

Texture2D

## Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## ContainsPixel(Texture2D, IntVector2)

Returns whether the given coordinates are within the bounds of the Texture2D.

```
public static bool ContainsPixel(this Texture2D texture, IntVector2 pixel)
```

## Parameters

`texture` Texture2D

`pixel` [IntVector2](#)

Returns

[bool](#)

## ContainsPixel(Texture2D, int, int)

Returns whether the coordinates `(x, y)` are within the bounds of the Texture2D.

```
public static bool ContainsPixel(this Texture2D texture, int x, int y)
```

Parameters

`texture` Texture2D

`x` [int](#)

`y` [int](#)

Returns

[bool](#)

## DeepCopy(Texture2D)

Creates a deep copy of the Texture2D.

```
public static Texture2D DeepCopy(this Texture2D texture)
```

Parameters

`texture` Texture2D

Returns

Texture2D

## ExtendCrop(Texture2D, IntRect)

Creates a deep copy of the Texture2D with the dimensions changed to fit the given [IntRect](#) by adding/removing rows/columns of pixels from the sides of the Texture2D.

```
public static Texture2D ExtendCrop(this Texture2D texture, IntRect newRect)
```

### Parameters

**texture** Texture2D

**newRect** [IntRect](#)

The coords of the new rect, relative to the coords of the old rect.

### Returns

Texture2D

### Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## ExtendCrop(Texture2D, in ExtendCropOptions)

Creates a deep copy of the Texture2D with rows/columns of pixels added to / removed from the sides.

```
public static Texture2D ExtendCrop(this Texture2D texture, in  
Texture2DExtensions.ExtendCropOptions options)
```

### Parameters

**texture** Texture2D

**options** [Texture2DExtensions.ExtendCropOptions](#)

### Returns

Texture2D

## Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## Exceptions

### [ArgumentException](#)

The resulting width is <= 0 or the resulting height is <= 0.

## Flip(Texture2D, FlipAxis)

Returns a deep copy of the Texture2D reflected across the given axis.

```
public static Texture2D Flip(this Texture2D texture, FlipAxis axis)
```

## Parameters

**texture** Texture2D

**axis** [FlipAxis](#)

## Returns

Texture2D

## Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## GetPixel(Texture2D, IntVector2)

Gets the pixel colour at the given coordinates.

```
public static Color GetPixel(this Texture2D texture, IntVector2 pixel)
```

## Parameters

**texture** Texture2D

`pixel` [IntVector2](#)

Returns

Color

## GetRect(Texture2D)

Returns an [IntRect](#) containing precisely the coordinates within the bounds of the Texture2D.

```
public static IntRect GetRect(this Texture2D texture)
```

Parameters

`texture` Texture2D

Returns

[IntRect](#)

## LoadFromFile(string)

Loads a Texture2D from the file at the given file path.

```
public static Texture2D LoadFromFile(string filePath)
```

Parameters

`filePath` [string](#) ↗

Returns

Texture2D

Remarks

Does not call Texture2D.Apply() on the returned Texture2D.

## Exceptions

### [FileNotFoundException](#)

The file path `filePath` does not exist.

## ReplaceColour(Texture2D, Color, Color, float)

Returns a deep copy of the Texture2D with all occurrences of the colour `toReplace` replaced with `replaceWith`.

```
public static Texture2D ReplaceColour(this Texture2D texture, Color toReplace, Color  
replaceWith, float tolerance = 0)
```

## Parameters

`texture` Texture2D

`toReplace` Color

`replaceWith` Color

`tolerance` [float](#)

How close a colour has to be to `toReplace` to be replaced.

## Returns

Texture2D

## Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## Exceptions

### [ArgumentOutOfRangeException](#)

`tolerance` is negative.

## Rotate(Texture2D, RotationAngle)

Returns a deep copy of the Texture2D rotated by the given angle.

```
public static Texture2D Rotate(this Texture2D texture, RotationAngle angle)
```

Parameters

**texture** Texture2D

**angle** [RotationAngle](#)

Returns

Texture2D

Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## Scale(Texture2D, int, int)

Returns a deep copy of the Texture2D scaled to size **newWidth** x **newHeight** using the nearest-neighbour algorithm.

```
public static Texture2D Scale(this Texture2D texture, int newWidth, int newHeight)
```

Parameters

**texture** Texture2D

**newWidth** [int](#)

**newHeight** [int](#)

Returns

Texture2D

Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## Exceptions

### [ArgumentOutOfRangeException](#)

`newWidth` is  $\leq 0$  or `newHeight` is  $\leq 0$ .

## Scale(Texture2D, float)

Calls [Scale\(Texture2D, float, float\)](#) with both scale factors equal to `scaleFactor`.

```
public static Texture2D Scale(this Texture2D texture, float scaleFactor)
```

## Parameters

`texture` Texture2D

`scaleFactor` [float](#)

## Returns

Texture2D

## Exceptions

### [ArgumentOutOfRangeException](#)

`scaleFactor` is  $\leq 0$ .

## Scale(Texture2D, float, float)

Returns a deep copy of the Texture2D with the width scaled by `xScaleFactor` and the height scaled by `yScaleFactor`, using the nearest-neighbour algorithm.

```
public static Texture2D Scale(this Texture2D texture, float xScaleFactor,
    float yScaleFactor)
```

## Parameters

`texture Texture2D`

`xScaleFactor float`

`yScaleFactor float`

Returns

Texture2D

Remarks

The width is multiplied by `xScaleFactor` then rounded. Similarly for the height.

Calls Texture2D.Apply() on the returned Texture2D.

Exceptions

[ArgumentOutOfRangeException](#)

`xScaleFactor` is <= 0 or `yScaleFactor` is <= 0.

[ArgumentException](#)

Rounding the scaled width results in a new width of 0 or rounding the scaled height results in a new height of 0.

## SetPixel(Texture2D, IntVector2, Color)

Sets the pixel colour at the given coordinates.

```
public static void SetPixel(this Texture2D texture, IntVector2 pixel, Color colour)
```

Parameters

`texture Texture2D`

`pixel IntVector2`

`colour Color`

# ToSprite(Texture2D)

Creates a Sprite from the Texture2D, using the FilterMode.Point filter mode.

```
public static Sprite ToSprite(this Texture2D texture)
```

## Parameters

**texture** Texture2D

## Returns

Sprite

## Remarks

This does not deep copy the Texture2D. This means that subsequent changes to the Texture2D will affect the Sprite once Texture2D.Apply() is called.

Does not call Texture2D.Apply() on the Texture2D.

# Struct Texture2DExtensions.ExtendCropOptions

Namespace: [PAC.Extensions](#)

Options for [ExtendCrop\(Texture2D, in ExtendCropOptions\)](#).

```
public struct Texture2DExtensions.ExtendCropOptions
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### bottom

How many rows of pixels to add to the bottom side of the texture.

```
public int bottom
```

#### Field Value

[int](#)

#### Remarks

Negative values will crop the texture.

### left

How many columns of pixels to add to the left side of the texture.

```
public int left
```

Field Value

[int](#)

Remarks

Negative values will crop the texture.

## right

How many columns of pixels to add to the right side of the texture.

```
public int right
```

Field Value

[int](#)

Remarks

Negative values will crop the texture.

## top

How many rows of pixels to add to the top side of the texture.

```
public int top
```

Field Value

[int](#)

Remarks

Negative values will crop the texture.

# Class TypeExtensions

Namespace: [PAC.Extensions](#)

```
public static class TypeExtensions
```

## Inheritance

[object](#) ← TypeExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### GetTypeOfRawGenericSuperclass(Type, Type)

Gets the concrete version of the given raw generic type that `subType` inherits from - e.g. `List<int>` inherits from the raw generic `IEnumerable<>`, with the concrete type returned by this method being `IEnumerable<int>`.

Throws an exception if `rawGeneric` is not a raw generic. For example, `List<>` is a raw generic, but `List<int>` is not.

Throws an exception if `subType` does not inherit from `rawGeneric`.

```
public static Type GetTypeOfRawGenericSuperclass(this Type subType, Type rawGeneric)
```

## Parameters

`subType` [Type](#)

`rawGeneric` [Type](#)

## Returns

[Type](#)

## GetValues<T>()

Gets all possible values of the [enum](#) type `T`.

```
public static T[] GetValues<T>() where T : Enum
```

Returns

`T[]`

Type Parameters

`T`

## IsAutoProperty(PropertyInfo)

Determines if the given property is an auto property - i.e. it has both a getter and setter with no body, for example

```
public string name { get; set; }
```

```
public static bool IsAutoProperty(this PropertyInfo property)
```

Parameters

`property`  [PropertyInfo](#)

Returns

[bool](#)

## IsGenericList(Type)

Determines if the given type is [List<T>](#) for some type `T`.

```
public static bool IsGenericList(this Type type)
```

Parameters

type [Type](#)

Returns

[bool](#)

## IsRawGeneric(Type)

Determines if the given type is a raw generic - e.g. `List<>` is a raw generic, but `List<int>` is not. Also returns false for non-generic types.

```
public static bool IsRawGeneric(this Type type)
```

Parameters

type [Type](#)

Returns

[bool](#)

## IsStruct(Type)

Determiens if the given type is a struct or not.

```
public static bool IsStruct(this Type type)
```

Parameters

type [Type](#)

Returns

[bool](#)

# IsSubclassOfRawGeneric(Type, Type)

Determines if the given type inherits from the given some concrete version of the given raw generic type - e.g. `List<int>` inherits from the raw generic `IEnumerable<>`.

Throws an exception if `rawGeneric` is not a raw generic. For example, `List<>` is a raw generic, but `List<int>` is not.

```
public static bool IsSubclassOfRawGeneric(this Type subType, Type rawGeneric)
```

Parameters

`subType` `Type`

`rawGeneric` `Type`

Returns

`bool`

# Class ValueTupleExtensions

Namespace: [PAC.Extensions](#)

```
public static class ValueTupleExtensions
```

## Inheritance

[object](#) ← ValueTupleExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### AsEnumerable<T>((T, T, T, T, T, T, T))

```
public static IEnumerable<T> AsEnumerable<T>(this (T, T, T, T, T, T, T) valueTuple)
```

#### Parameters

**valueTuple** (T, T, T, T, T, T, T)

#### Returns

[IEnumerable](#)<T>

#### Type Parameters

T

### AsEnumerable<T>((T, T, T, T, T, T))

```
public static IEnumerable<T> AsEnumerable<T>(this (T, T, T, T, T, T) valueTuple)
```

Parameters

**valueTuple** (T, T, T, T, T, T)

Returns

[IEnumerable](#)<T>

Type Parameters

T

**AsEnumerable<T>((T, T, T, T, T))**

```
public static IEnumerable<T> AsEnumerable<T>(this (T, T, T, T, T) valueTuple)
```

Parameters

**valueTuple** (T, T, T, T, T)

Returns

[IEnumerable](#)<T>

Type Parameters

T

**AsEnumerable<T>((T, T, T, T))**

```
public static IEnumerable<T> AsEnumerable<T>(this (T, T, T, T) valueTuple)
```

Parameters

**valueTuple** (T, T, T, T)

Returns

[IEnumerable](#)<T>

Type Parameters

T

AsEnumerable<T>((T, T, T))

```
public static IEnumerable<T> AsEnumerable<T>(this (T, T, T) valueTuple)
```

Parameters

valueTuple (T, T, T)

Returns

[IEnumerable](#)<T>

Type Parameters

T

AsEnumerable<T>((T, T))

```
public static IEnumerable<T> AsEnumerable<T>(this (T, T) valueTuple)
```

Parameters

valueTuple (T, T)

Returns

[IEnumerable](#)<T>

Type Parameters

T

## AsEnumerable<T>(ValueTuple<T>)

```
public static IEnumerable<T> AsEnumerable<T>(this ValueTuple<T> valueTuple)
```

Parameters

**valueTuple** [ValueTuple](#)<T>

Returns

[IEnumerable](#)<T>

Type Parameters

T

## Index<T>((T, T, T, T, T, T, T), int)

Returns the element at the given index in the tuple.

```
public static T Index<T>(this (T, T, T, T, T, T, T) valueTuple, int index)
```

Parameters

**valueTuple** (T, T, T, T, T, T, T)

**index** [int](#)

Returns

T

Type Parameters

T

## Index<T>((T, T, T, T, T, T), int)

Returns the element at the given index in the tuple.

```
public static T Index<T>(this (T, T, T, T, T, T) valueTuple, int index)
```

Parameters

**valueTuple** (T, T, T, T, T, T)

**index** [int](#)

Returns

T

Type Parameters

T

## Index<T>((T, T, T, T, T), int)

Returns the element at the given index in the tuple.

```
public static T Index<T>(this (T, T, T, T, T) valueTuple, int index)
```

Parameters

**valueTuple** (T, T, T, T, T)

**index** [int](#)

Returns

T

Type Parameters

T

## Index<T>((T, T, T, T), int)

Returns the element at the given index in the tuple.

```
public static T Index<T>(this (T, T, T, T) valueTuple, int index)
```

Parameters

valueTuple (T, T, T, T)

index [int](#)

Returns

T

Type Parameters

T

## Index<T>((T, T, T), int)

Returns the element at the given index in the tuple.

```
public static T Index<T>(this (T, T, T) valueTuple, int index)
```

Parameters

valueTuple (T, T, T)

index [int](#)

Returns

T

Type Parameters

T

## Index<T>((T, T), int)

Returns the element at the given index in the tuple.

```
public static T Index<T>(this (T, T) valueTuple, int index)
```

Parameters

valueTuple (T, T)

index [int](#)

Returns

T

Type Parameters

T

## Index<T>(ValueTuple<T>, int)

Returns the element at the given index in the tuple.

```
public static T Index<T>(this ValueTuple<T> valueTuple, int index)
```

Parameters

valueTuple [ValueTuple](#)<T>

index [int](#)

Returns

T

Type Parameters

T



# Namespace PAC.Files

## Classes

### [File](#)

A class to represent a single Pixel Art Creator file.

### [File.JsonConverter](#)

### [FileManager](#)

### [FileTab](#)

### [FileTabsManager](#)

### [ImageEditManager](#)

# Class File

Namespace: [PAC.Files](#)

A class to represent a single Pixel Art Creator file.

```
public class File
```

## Inheritance

[object](#) ← File

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### File(File)

Creates a deep copy of the File.

```
public File(File file)
```

#### Parameters

`file` [File](#)

### File(string, int, int)

Creates a blank file.

```
public File(string name, int width, int height)
```

## Parameters

`name` [string](#)

`width` [int](#)

`height` [int](#)

## Properties

### height

`public int height { get; }`

Property Value

[int](#)

### keyFrameIndices

The frame indices at which some layer has a key frame.

`public int[] keyFrameIndices { get; }`

Property Value

[int](#)[]

### layers

`public List<Layer> layers { get; }`

Property Value

[List](#)<[Layer](#)>

## liveRender

```
public Texture2D liveRender { get; }
```

Property Value

Texture2D

## mostRecentSavePath

```
public string mostRecentSavePath { get; }
```

Property Value

[string](#)

## name

```
public string name { get; set; }
```

Property Value

[string](#)

## numOfFrames

The number of frames the animation lasts for.

```
public int numOfFrames { get; set; }
```

Property Value

[int](#)

## rect

```
public IntRect rect { get; }
```

Property Value

[IntRect](#)

## savedSinceLastEdit

```
public bool savedSinceLastEdit { get; }
```

Property Value

[bool](#)

## tiles

All the tile objects currently in the file.

```
public Tile[] tiles { get; }
```

Property Value

[Tile\[\]](#)

## width

```
public int width { get; }
```

Property Value

[int](#)

# Methods

## AddLayer(Layer)

Adds the given layer on top of all existing layers.

```
public void AddLayer(Layer layer)
```

### Parameters

layer [Layer](#)

## AddLayer(Layer, int)

Adds the given layer at the given index. Throws an error if the layer is already in the file.

```
public void AddLayer(Layer layer, int index)
```

### Parameters

layer [Layer](#)

index [int](#)

## AddLayers(params Layer[])

Adds the layers on top of all existing layers, retaining the order they have in the array - i.e. first layer in the array is the one on top, etc.

```
public void AddLayers(params Layer[] layers)
```

### Parameters

layers [Layer\[\]](#)

## AddNormalLayer()

Adds a blank normal layer on top of all existing layers.

```
public void AddNormalLayer()
```

## AddNormalLayer(int)

Adds a blank normal layer at the given index.

```
public void AddNormalLayer(int index)
```

Parameters

**index** [int](#)

## AddNormalLayer(Texture2D, int)

Adds a normal layer at the given index with the given texture.

```
public void AddNormalLayer(Texture2D texture, int index = 0)
```

Parameters

**texture** Texture2D

**index** [int](#)

## AddTile(File, IntVector2, int)

Adds the given file as a tile.

```
public void AddTile(File file, IntVector2 bottomLeft, int lowestLayer)
```

Parameters

**file** [File](#)

`bottomLeft` [IntVector2](#)

`lowestLayer` [int](#)

## AddTile(Tile)

Adds the tile to the file.

```
public void AddTile(Tile tile)
```

### Parameters

`tile` [Tile](#)

## AddTileLayer()

Adds a blank tile layer on top of all existing layers.

```
public void AddTileLayer()
```

## AddTileLayer(int)

Adds a blank tile layer at the given index.

```
public void AddTileLayer(int index)
```

### Parameters

`index` [int](#)

## ContainsTile(Tile)

```
public bool ContainsTile(Tile tile)
```

Parameters

[tile](#) [Tile](#)

Returns

[bool](#) ↗

## DeepCopy()

Creates a deep copy of the File.

```
public File DeepCopy()
```

Returns

[File](#)

## ExportAnimation(string)

Exports each frame of the animation and puts them in a folder. The file path specifies the folder name and location, and the file extension to export each frame as.

```
public bool ExportAnimation(string filePath)
```

Parameters

[filePath](#) [string](#) ↗

Returns

[bool](#) ↗

## ExportFrame(int, string)

Exports the frame to the file path, according to the file extension.

```
public bool ExportFrame(int frame, string filePath)
```

Parameters

frame [int](#)

filePath [string](#)

Returns

[bool](#)

## Extend(int, int, int, int)

Extends the dimensions of the file in each direction by the given amounts.

```
public void Extend(int left, int right, int up, int down)
```

Parameters

left [int](#)

right [int](#)

up [int](#)

down [int](#)

## Flip(FlipAxis)

Flips the file.

```
public void Flip(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

## ImportImage(string)

Adds the image at the file path as a new layer.

```
public void ImportImage(string filePath)
```

Parameters

filePath [string](#)

## ImportLayers(params Layer[])

Adds deep copies of the layers on top of all existing layers, retaining the order they have in the array - i.e. first layer in the array is the one on top, etc.

```
public void ImportLayers(params Layer[] layers)
```

Parameters

layers [Layer](#)[]

## IndexOfLayer(Layer)

Gets the index of the layer within the file.

```
public int IndexOfLayer(Layer layer)
```

Parameters

layer [Layer](#)

Returns

[int](#)

The index of the layer, or -1 if the layer is not in the file.

## IsBlank()

Returns true if and only if all pixels on all layers are completely transparent.

```
public bool IsBlank()
```

Returns

[bool](#)

## MoveLayer(Layer, int)

Moves the layer to indexToMoveTo. Throws an error if the layer is not in the file.

```
public void MoveLayer(Layer layerToMove, int indexToMoveTo)
```

Parameters

layerToMove [Layer](#)

indexToMoveTo [int](#)

## MoveLayer(int, int)

Moves the layer at indexToMove to indexToMoveTo.

```
public void MoveLayer(int layerToMove, int indexToMoveTo)
```

Parameters

layerToMove [int](#)

indexToMoveTo [int](#)

## OpenFile(string)

Opens the file according to its file extension.

```
public static File OpenFile(string filePath)
```

Parameters

**filePath** [string](#)

Returns

[File](#)

## OpenImage(string)

Opens the file at the file path if it is an image file type, e.g. PNG or JPEG.

```
public static File OpenImage(string filePath)
```

Parameters

**filePath** [string](#)

Returns

[File](#)

## OpenJPEG(string)

Opens the JPEG/JPG file. Throws an error if the file is not a JPEG/JPG file.

```
public static File OpenJPEG(string filePath)
```

Parameters

**filePath** [string](#)

Returns

[File](#)

## OpenPAC(string)

Opens the PAC file. Throws an error if the file is not a PAC file.

```
public static File OpenPAC(string filePath)
```

Parameters

`filePath` [string](#)

Returns

[File](#)

## OpenPNG(string)

Opens the PNG file. Throws an error if the file is not a PNG file.

```
public static File OpenPNG(string filePath)
```

Parameters

`filePath` [string](#)

Returns

[File](#)

## RemoveAllLayers()

Removes all layers, then adds a blank normal layer.

```
public void RemoveAllLayers()
```

## RemoveLayer(Layer)

Removes the layer from the file. Throws an error if the layer is not in the file.

```
public void RemoveLayer(Layer layer)
```

Parameters

layer [Layer](#)

## RemoveLayer(int)

Removes the layer at the given index.

```
public void RemoveLayer(int layer)
```

Parameters

layer [int](#)

## RemoveLayers(Layer[])

Removes the given layers from the file. Throws an error if a layer is not in the file.

```
public void RemoveLayers(Layer[] layers)
```

Parameters

layers [Layer](#)[]

## RemoveLayers(int[])

Removes the layers at the given indices.

```
public void RemoveLayers(int[] layers)
```

Parameters

`layers int[]`

## RemoveTile(Tile)

Removes the tile. Throws an error if the tile is not in the file.

```
public void RemoveTile(Tile tile)
```

Parameters

`tile Tile`

## Render(int)

Renders all layers down to a Texture2D. Does not apply the texture.

```
public Texture2D Render(int frame)
```

Parameters

`frame int`

Returns

Texture2D

## RenderLayers(IEnumerable<int>, int)

Renders the layers at the given layer indices down to a Texture2D. Does not apply the texture.

```
public Texture2D RenderLayers(IEnumerable<int> layerIndices, int frame)
```

Parameters

`layerIndices` [IEnumerable<int>](#)

The layer indices in the order you want them to be rendered, from highest layer (so lowest index) to lowest.

`frame` [int](#)

Returns

`Texture2D`

## RenderLayers(int, int, int, bool)

Renders all the layers with index between `lowestLayer` and `highestLayer` (inclusive iff `inclusive == true`) down to a `Texture2D`. `highestLayer` and `lowestLayer` are automatically clamped to be in the valid range of layers. Does not apply the texture.

```
public Texture2D RenderLayers(int highestLayer, int lowestLayer, int frame, bool inclusive = true)
```

Parameters

`highestLayer` [int](#)

The higher layer (so lower index).

`lowestLayer` [int](#)

The lower layer (so higher index).

`frame` [int](#)

`inclusive` [bool](#)

Returns

`Texture2D`

## RenderLayersAbove(int, int, bool)

Renders all layers that appear above the given layer (includes the layer iff inclusive == true) down to a Texture2D. Returns a blank texture if there are none. Does not apply the texture.

```
public Texture2D RenderLayersAbove(int layer, int frame, bool inclusive = false)
```

Parameters

layer [int](#)

frame [int](#)

inclusive [bool](#)

Returns

Texture2D

## RenderLayersBelow(int, int, bool)

Renders all layers that appear strictly beneath the given layer (includes the layer iff inclusive == true) down to a Texture2D. Returns a blank texture if there are none. Does not apply the texture.

```
public Texture2D RenderLayersBelow(int layer, int frame, bool inclusive = false)
```

Parameters

layer [int](#)

frame [int](#)

inclusive [bool](#)

Returns

Texture2D

## RenderPixel(IntVector2, IEnumerable<int>, int)

Renders the colour of the pixel on the layers at the given layer indices. Throws an error if there are no layer indices.

```
public Color RenderPixel(IntVector2 pixel, IEnumerable<int> layerIndices, int frame)
```

Parameters

`pixel` [IntVector2](#)

`layerIndices` [IEnumerable](#)<[int](#)>

`frame` [int](#)

Returns

Color

## RenderPixel(IntVector2, int)

Renders the colour of the given pixel.

```
public Color RenderPixel(IntVector2 pixel, int frame)
```

Parameters

`pixel` [IntVector2](#)

`frame` [int](#)

Returns

Color

## RenderPixel(IntVector2, int, int, int, bool)

```
public Color RenderPixel(IntVector2 pixel, int highestLayer, int lowestLayer, int
```

```
frame, bool inclusive = true)
```

## Parameters

pixel [IntVector2](#)

highestLayer [int](#)

lowestLayer [int](#)

frame [int](#)

inclusive [bool](#)

## Returns

Color

## RenderPixel(int, int, IEnumerable<int>, int)

Renders the colour of pixel (x, y) on the layers at the given layer indices. Throws an error if there are no layer indices.

```
public Color RenderPixel(int x, int y, IEnumerable<int> layerIndices, int frame)
```

## Parameters

x [int](#)

y [int](#)

layerIndices [IEnumerable](#)<[int](#)>

frame [int](#)

## Returns

Color

## RenderPixel(int, int)

Renders the colour of the given pixel.

```
public Color RenderPixel(int x, int y, int frame)
```

Parameters

x [int](#)

y [int](#)

frame [int](#)

Returns

Color

## RenderPixel(int, int, int, int, int, bool)

```
public Color RenderPixel(int x, int y, int highestLayer, int lowestLayer, int frame,  
bool inclusive = true)
```

Parameters

x [int](#)

y [int](#)

highestLayer [int](#)

lowestLayer [int](#)

frame [int](#)

inclusive [bool](#)

Returns

Color

## ReplaceLayer(Layer, Layer)

Replaces the layer with the given layer. Throws an error if the layer to replace is not in the file, or if the layer to replace with is already in the file.

```
public void ReplaceLayer(Layer layerToReplace, Layer layerToReplaceWith)
```

### Parameters

layerToReplace [Layer](#)

layerToReplaceWith [Layer](#)

## ReplaceLayer(int, Layer)

Replaces the layer at the given index with the given layer. Throws an error if the layer to replace with is already in the file.

```
public void ReplaceLayer(int layerToReplace, Layer layerToReplaceWith)
```

### Parameters

layerToReplace [int](#)

layerToReplaceWith [Layer](#)

## Rotate(RotationAngle)

Rotates the file. Rotation is clockwise.

```
public void Rotate(RotationAngle angle)
```

### Parameters

angle [RotationAngle](#)

## SaveAsPAC(string)

Saves the file as a PAC file at the given file path. Throws an error if the file is not a PAC file.

```
public void SaveAsPAC(string filePath)
```

Parameters

filePath [string](#)

## SavePAC()

Saves the file as a PAC file at the location it was most recently saved at. Throws an error if the file hasn't been saved before.

```
public void SavePAC()
```

## Scale(int, int)

Resizes the file to the new width and height.

```
public void Scale(int newWidth, int newHeight)
```

Parameters

newWidth [int](#)

newHeight [int](#)

## Scale(float)

Resizes the dimensions of the file by the scale factor.

```
public void Scale(float scaleFactor)
```

Parameters

scaleFactor [float](#)

## Scale(float, float)

Resizes the dimensions of the file by the scale factors.

```
public void Scale(float xScaleFactor, float yScaleFactor)
```

Parameters

xScaleFactor [float](#)

yScaleFactor [float](#)

## SetLiveRenderFrame(int)

Sets the frame that the live render will be for.

```
public void SetLiveRenderFrame(int frame)
```

Parameters

frame [int](#)

## SubscribeToOnPixelsChanged(UnityAction<IEnumerable<IntVector2>, Layer[], int[]>)

Event invoked when some pixels have been changed on a layer. Passes the pixels, layers and frames that were affected.

```
public void SubscribeToOnPixelsChanged(UnityAction<IEnumerable<IntVector2>, Layer[], int[]> call)
```

Parameters

call [UnityAction<IEnumerable<IntVector2>, Layer\[\], int\[\]>](#)

## SubscribeToOnSavedSinceEditChanged(UnityAction)

Event invoked when the a change to file is made or when the file is saved.

```
public void SubscribeToOnSavedSinceEditChanged(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToOnTileAdded(UnityAction)

Event invoked when a tile is added to the file.

```
public void SubscribeToOnTileAdded(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToOnTileRemoved(UnityAction)

Event invoked when a tile is removed from the file.

```
public void SubscribeToOnTileRemoved(UnityAction call)
```

Parameters

`call` UnityAction

# Class File.JsonConverter

Namespace: [PAC.Files](#)

```
public class File.JsonConverter : JsonConversion.JsonConverter<File,  
JsonData.Object>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<File, jsonData.Object>](#) ← [File.JsonConverter](#)

## Inherited Members

[JsonConversion.JsonConverter<File, jsonData.Object>.ToJson\(File\)](#) ,  
[JsonConversion.JsonConverter<File, jsonData.Object>.FromJson\(jsonData.Object\)](#) ,  
[JsonConversion.JsonConverter<File, jsonData.Object>.FromJson\(jsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Constructors

### JsonConverter(SemanticVersion)

```
public JsonConverter(SemanticVersion fromJsonFileVersionVersion)
```

## Parameters

[fromJsonFileVersionVersion SemanticVersion](#)

## Methods

### FromJson(Object)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override File FromJson(JsonData.Object jsonData)
```

Parameters

jsonData [JsonData.Object](#)

Returns

[File](#)

## ToJson(File)

Attempts to convert the C# object into JSON data.

```
public override JsonData.Object ToJson(File file)
```

Parameters

file [File](#)

Returns

[JsonData.Object](#)

# Class FileManager

Namespace: [PAC.Files](#)

```
public class FileManager : MonoBehaviour
```

## Inheritance

[object](#) ← FileManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### defaultFileName

```
public const string defaultFileName = "Untitled Image"
```

Field Value

[string](#)

### previousFile

```
public File previousFile
```

Field Value

[File](#)

## Properties

### currentFile

```
public File currentFile { get; }
```

Property Value

[File](#)

## currentFileIndex

```
public int currentFileIndex { get; }
```

Property Value

[int](#)

## files

```
public List<File> files { get; }
```

Property Value

[List](#)<[File](#)>

## Methods

### AddFile(File)

```
public bool AddFile(File file)
```

Parameters

`file` [File](#)

Returns

[bool](#)

## CloseFile(File)

Closes the file, with no checks for unsaved changes.

```
public bool CloseFile(File file)
```

Parameters

[file](#) [File](#)

Returns

[bool](#)

## CloseFile(int)

Closes the file, with no checks for unsaved changes.

```
public File CloseFile(int fileIndex)
```

Parameters

[fileIndex](#) [int](#)

Returns

[File](#)

## ExportAnimation(int, string)

```
public bool ExportAnimation(int fileIndex, string filePath)
```

Parameters

`fileIndex` [int ↗](#)

`filePath` [string ↗](#)

Returns

[bool ↗](#)

## ExportCurrentAnimation(string)

```
public bool ExportCurrentAnimation(string filePath)
```

Parameters

`filePath` [string ↗](#)

Returns

[bool ↗](#)

## ExportCurrentAnimationDialog()

```
public void ExportCurrentAnimationDialog()
```

## ExportCurrentFrame(int, string)

```
public bool ExportCurrentFrame(int frameIndex, string filePath)
```

Parameters

`frameIndex` [int ↗](#)

`filePath` [string ↗](#)

Returns

[bool](#)

## ExportCurrentFrameDialog()

```
public void ExportCurrentFrameDialog()
```

## ExportFrame(int, int, string)

```
public bool ExportFrame(int frameIndex, int fileIndex, string filePath)
```

### Parameters

frameIndex [int](#)

fileIndex [int](#)

filePath [string](#)

### Returns

[bool](#)

## ImportDialog()

```
public void ImportDialog()
```

## NewFile(int, int)

```
public void NewFile(int width, int height)
```

### Parameters

width [int](#)

height [int](#)

## OpenFile(File)

`public bool OpenFile(File file)`

Parameters

[file](#) [File](#)

Returns

[bool](#)

## OpenFile(string)

`public bool OpenFile(string filePath)`

Parameters

[filePath](#) [string](#)

Returns

[bool](#)

## OpenFileDialog()

`public void OpenFileDialog()`

## ReloadFile()

`public void ReloadFile()`

## SaveAsCurrentFile(string)

```
public void SaveAsCurrentFile(string filePath)
```

### Parameters

filePath [string](#)

## SaveAsCurrentFileDialog()

```
public void SaveAsCurrentFileDialog()
```

## SaveAsFile(File, string)

```
public void SaveAsFile(File file, string filePath)
```

### Parameters

file [File](#)

filePath [string](#)

## SaveAsFile(int, string)

```
public void SaveAsFile(int fileIndex, string filePath)
```

### Parameters

fileIndex [int](#)

filePath [string](#)

## SaveAsFileDialog(File)

```
public void SaveAsFileDialog(File file)
```

Parameters

`file` [File](#)

## SaveCurrentFileDialog()

```
public void SaveCurrentFileDialog()
```

## SaveFileDialog(File)

```
public void SaveFileDialog(File file)
```

Parameters

`file` [File](#)

## SubscribeToFileSwitched(UnityAction)

```
public void SubscribeToFileSwitched(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToFilesChanged(UnityAction)

```
public void SubscribeToFilesChanged(UnityAction call)
```

Parameters

**call** UnityAction

## SwitchToFile(int)

```
public void SwitchToFile(int fileIndex)
```

Parameters

fileIndex [int](#)

## TryCloseFile(File)

Will close the file if there are no unsaved changes. Otherwise it will open a dialog box asking if you want to save.

```
public bool TryCloseFile(File file)
```

Parameters

file [File](#)

Returns

[bool](#)

## TryCloseFile(int)

Will close the file if there are no unsaved changes. Otherwise it will open a dialog box asking if you want to save.

```
public bool TryCloseFile(int fileIndex)
```

Parameters

fileIndex [int](#)

Returns

bool ↗

# Class FileTab

Namespace: [PAC.Files](#)

```
public class FileTab : MonoBehaviour
```

## Inheritance

[object](#) ← FileTab

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

closed

```
public bool closed { get; set; }
```

Property Value

[bool](#)

file

```
public File file { get; }
```

Property Value

[File](#)

selected

```
public bool selected { get; }
```

Property Value

[bool](#)

## tileToggle

```
public UIToggleButton tileToggle { get; }
```

Property Value

[UIToggleButton](#)

## Methods

### SetFile(File)

```
public void SetFile(File file)
```

Parameters

[file](#) [File](#)

### SubscribeToClose(UnityAction)

```
public void SubscribeToClose(UnityAction call)
```

Parameters

[call](#) [UnityAction](#)

### SubscribeToNameChange(UnityAction)

```
public void SubscribeToNameChange(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToSelect(UnityAction)

```
public void SubscribeToSelect(UnityAction call)
```

Parameters

`call` UnityAction

# Class FileTabsManager

Namespace: [PAC.Files](#)

```
public class FileTabsManager : MonoBehaviour
```

## Inheritance

[object](#) ↗ ← FileTabsManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### selectedFile

```
public File selectedFile { get; }
```

Property Value

[File](#)

### selectedFileIndex

```
public int selectedFileIndex { get; }
```

Property Value

[int](#) ↗

## Methods

### CloseFile()

```
public void CloseFile()
```

## OnFilesChanged()

```
public void OnFilesChanged()
```

## SelectFile()

```
public void SelectFile()
```

## SubscribeToOnFilesChanged(UnityAction)

```
public void SubscribeToOnFilesChanged(UnityAction call)
```

### Parameters

`call` UnityAction

# Class ImageEditManager

Namespace: [PAC.Files](#)

```
public class ImageEditManager : MonoBehaviour
```

## Inheritance

[object](#) ← ImageEditManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### ExtendFile(int, int, int, int)

```
public void ExtendFile(int left, int right, int up, int down)
```

#### Parameters

left [int](#)

right [int](#)

up [int](#)

down [int](#)

### ExtendFileDown(int)

```
public void ExtendFileDown(int amount)
```

#### Parameters

amount [int](#)

## ExtendFileLeft(int)

```
public void ExtendFileLeft(int amount)
```

### Parameters

amount [int](#)

## ExtendFileRight(int)

```
public void ExtendFileRight(int amount)
```

### Parameters

amount [int](#)

## ExtendFileUp(int)

```
public void ExtendFileUp(int amount)
```

### Parameters

amount [int](#)

## FlipFile(FlipAxis)

```
public void FlipFile(FlipAxis axis)
```

### Parameters

axis [FlipAxis](#)

## FlipFileX()

```
public void FlipFileX()
```

## FlipFileY()

```
public void FlipFileY()
```

## FlipSelectedLayers(FlipAxis)

```
public void FlipSelectedLayers(FlipAxis axis)
```

### Parameters

axis [FlipAxis](#)

## FlipSelectedLayersX()

```
public void FlipSelectedLayersX()
```

## FlipSelectedLayersY()

```
public void FlipSelectedLayersY()
```

## RotateFile(RotationAngle)

```
public void RotateFile(RotationAngle angle)
```

### Parameters

angle [RotationAngle](#)

## RotateSelectedLayers(RotationAngle)

```
public void RotateSelectedLayers(RotationAngle angle)
```

### Parameters

angle [RotationAngle](#)

## RotateSelectedLayers180()

```
public void RotateSelectedLayers180()
```

## RotateSelectedLayers90()

```
public void RotateSelectedLayers90()
```

## RotateSelectedLayersMinus90()

```
public void RotateSelectedLayersMinus90()
```

## ScaleFile(int, int)

```
public void ScaleFile(int newWidth, int newHeight)
```

### Parameters

newWidth [int](#)

newHeight [int](#)

## ScaleFile(float)

```
public void ScaleFile(float scaleFactor)
```

Parameters

scaleFactor [float](#)

## ScaleFile(float, float)

```
public void ScaleFile(float xScaleFactor, float yScaleFactor)
```

Parameters

xScaleFactor [float](#)

yScaleFactor [float](#)

## SubscribeToEdit(UnityAction)

```
public void SubscribeToEdit(UnityAction call)
```

Parameters

call UnityAction

## SubscribeToImageSizeChanged(UnityAction)

```
public void SubscribeToImageSizeChanged(UnityAction call)
```

Parameters

call UnityAction

# Namespace PAC.ImageEditing

## Classes

### [FloodFill](#)

Handles flood-filling pixel art images.

### [Outline](#)

Handles creating pixel art outlines.

### [Texture2DCreator](#)

Provides methods to create Texture2Ds.

## Structs

### [Outline.Options](#)

Defines how the outline in [DrawOutline\(Texture2D, Color, in Options\)](#) will look.

## Enums

### [Outline.Options.OutlineType](#)

# Class FloodFill

Namespace: [PAC.ImageEditing](#)

Handles flood-filling pixel art images.

```
public static class FloodFill
```

## Inheritance

[object](#) ← FloodFill

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### GetPixelsToFill(Texture2D, IntVector2, int)

Returns the largest connected (in terms of being adjacent (left/right/up/down)) set containing `startPoint` where all pixels have the same colour.

```
public static IEnumerable<IntVector2> GetPixelsToFill(Texture2D texture, IntVector2  
startPoint, int maxNumOfIterations = 1000000)
```

## Parameters

`texture` Texture2D

`startPoint` [IntVector2](#)

`maxNumOfIterations` [int](#)

After this many pixels have been enumerated, the method will stop. Useful to prevent huge frame drops when filling large areas.

## Returns

[IEnumerable](#) <IntVector2>

# Class Outline

Namespace: [PAC.ImageEditing](#)

Handles creating pixel art outlines.

```
public static class Outline
```

## Inheritance

[object](#) ← Outline

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### DrawOutline(Texture2D, Color, in Options)

Returns a deep copy of the Texture2D with an outline around the non-transparent pixels.

```
public static Texture2D DrawOutline(Texture2D texture, Color outlineColour, in  
Outline.Options outlineOptions)
```

#### Parameters

**texture** Texture2D

**outlineColour** Color

**outlineOptions** [Outline.Options](#)

#### Returns

Texture2D

#### Remarks

Calls Texture2D.Apply() on the returned Texture2D.

# Struct Outline.Options

Namespace: [PAC.ImageEditing](#)

Defines how the outline in [DrawOutline\(Texture2D, Color, in Options\)](#) will look.

```
public struct Outline.Options
```

## Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### includeBottomLeft

Whether to draw the outline on the bottom-left edges.

```
public bool includeBottomLeft
```

#### Field Value

[bool](#)

### includeBottomMiddle

Whether to draw the outline on the bottom-middle edges.

```
public bool includeBottomMiddle
```

#### Field Value

[bool](#)

## includeBottomRight

Whether to draw the outline on the bottom-right edges.

```
public bool includeBottomRight
```

Field Value

[bool](#)

## includeMiddleLeft

Whether to draw the outline on the middle-left edges.

```
public bool includeMiddleLeft
```

Field Value

[bool](#)

## includeMiddleRight

Whether to draw the outline on the middle-right edges.

```
public bool includeMiddleRight
```

Field Value

[bool](#)

## includeTopLeft

Whether to draw the outline on the top-left edges.

```
public bool includeTopLeft
```

Field Value

[bool](#) ↗

## includeTopMiddle

Whether to draw the outline on the top-middle edges.

```
public bool includeTopMiddle
```

Field Value

[bool](#) ↗

## includeTopRight

Whether to draw the outline on the top-right edges.

```
public bool includeTopRight
```

Field Value

[bool](#) ↗

## outlineType

```
public Outline.Options.OutlineType outlineType
```

Field Value

[Outline.Options.OutlineType](#)

# Methods

## EnumerateDirectionsToInclude()

Iterates over the directions associated with [includeTopLeft](#) etc for those that are [true](#).

```
public readonly IEnumerable<IntVector2> EnumerateDirectionsToInclude()
```

Returns

[IEnumerable](#)<[IntVector2](#)>

# Enum Outline.Options.OutlineType

Namespace: [PAC.ImageEditing](#)

```
public enum Outline.Options.OutlineType
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

**Inside = 0**

The outline will be drawn on non-transparent pixels that are adjacent to transparent pixels.

**Outside = 1**

The outline will be drawn on transparent pixels that are adjacent to non-transparent pixels.

# Class Texture2D.Creator

Namespace: [PAC.ImageEditing](#)

Provides methods to create Texture2Ds.

```
public static class Texture2D.Creator
```

## Inheritance

[object](#) ← Texture2D.Creator

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### HSLHueSaturationGrid(int, int)

Creates a Texture2D of size `width` x `height`, where the colour of a pixel is determined in HSL as follows:

- **Hue**: linearly interpolated based on the x coord, such that the left-most pixels have hue 0 and the texture goes through all hues exactly once.
- **Saturation**: linearly interpolated based on the y coord, such that the top-most pixels have saturation 1 and the bottom-most pixels have saturation 0.
- **Lightness**: 0.5.

```
public static Texture2D HSLHueSaturationGrid(int width, int height)
```

## Parameters

`width` [int](#)

`height` [int](#)

## Returns

## Texture2D

### Remarks

Calls Texture2D.Apply() on the returned Texture2D.

## Solid(int, int, Color)

Creates a Texture2D of size `width x height`, filled with the given colour.

```
public static Texture2D Solid(int width, int height, Color colour)
```

### Parameters

`width` [int](#)

`height` [int](#)

`colour` Color

### Returns

Texture2D

### Remarks

[Solid\(int, int, Color32\)](#) is faster.

Calls Texture2D.Apply() on the returned Texture2D.

### Exceptions

[ArgumentOutOfRangeException](#)

`width` or `height` is  $\leq 0$ .

## Solid(int, int, Color32)

Creates a Texture2D of size `width x height`, filled with the given colour.

```
public static Texture2D Solid(int width, int height, Color32 colour)
```

## Parameters

width [int](#)

height [int](#)

colour Color32

## Returns

Texture2D

## Remarks

This is faster than [Solid\(int, int, Color\)](#).

Calls Texture2D.Apply() on the returned Texture2D.

## Exceptions

[ArgumentOutOfRangeException](#)

width or height is <= 0.

## Transparent(int, int)

Creates a transparent (alpha 0) Texture2D of size width x height.

```
public static Texture2D Transparent(int width, int height)
```

## Parameters

width [int](#)

height [int](#)

## Returns

## Texture2D

### Remarks

Calls Texture2D.Apply() on the returned Texture2D.

### Exceptions

#### [ArgumentOutOfRangeException](#)

`width` or `height` is <= 0.

## TransparentCheckerboardBackground(int, int)

Creates a checkerboard Texture2D to act as the background for transparent pixels for a `width` x `height` image.

```
public static Texture2D TransparentCheckerboardBackground(int width, int height)
```

### Parameters

`width` [int](#)

`height` [int](#)

### Returns

Texture2D

### Remarks

The checkerboard texture will have dimensions  $2 * \text{width} \times 2 * \text{height}$ .

Calls Texture2D.Apply() on the returned Texture2D.

# Namespace PAC.Input

## Classes

[InputSystem](#)

[InputTarget](#)

[Keyboard](#)

[KeyboardTarget](#)

[Mouse](#)

[MouseTarget](#)

[Tooltip](#)

## Enums

[CursorState](#)

[MouseButton](#)

[MouseTargetDeselectMode](#)

[MouseTargetState](#)

# Enum CursorState

Namespace: [PAC.Input](#)

```
public enum CursorState
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

CrossArrows = 7

CurrentTool = -1

EyeDropper = 5

Grab = 12

Hover = 2

Invisible = 4

LeftRightArrow = 9

LinearGradient = 30

MagicWand = 11

MagnifyingGlass = 10

Normal = 1

Press = 3

RadialGradient = 31

SelectionEllipse = 101

SelectionRectangle = 100

Text = 6

Unspecified = 0

UpDownArrow = 8

# Class InputSystem

Namespace: [PAC.Input](#)

```
public class InputSystem : MonoBehaviour
```

## Inheritance

[object](#) ← InputSystem

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### timeBetweenKeySpam

```
public float timeBetweenKeySpam
```

Field Value

[float](#)

### timeUntilKeySpam

```
public float timeUntilKeySpam
```

Field Value

[float](#)

## Properties

### globalInputTarget

```
public InputTarget globalInputTarget { get; }
```

Property Value

[InputTarget](#)

## globalKeyboardTarget

```
public KeyboardTarget globalKeyboardTarget { get; }
```

Property Value

[KeyboardTarget](#)

## globalMouseTarget

```
public MouseTarget globalMouseTarget { get; }
```

Property Value

[MouseTarget](#)

## hasInputTarget

```
public bool hasInputTarget { get; }
```

Property Value

[bool](#)

## inputTarget

```
public InputTarget inputTarget { get; }
```

Property Value

[InputTarget](#)

## keyboard

```
public Keyboard keyboard { get; }
```

Property Value

[Keyboard](#)

## mouse

```
public Mouse mouse { get; }
```

Property Value

[Mouse](#)

## Methods

### LockForAFrame()

```
public void LockForAFrame()
```

### SubscribeToGlobalKeyboard(UnityAction)

```
public void SubscribeToGlobalKeyboard(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToGlobalLeftClick(UnityAction)

```
public void SubscribeToGlobalLeftClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToGlobalMouseScroll(UnityAction)

```
public void SubscribeToGlobalMouseScroll(UnityAction call)
```

Parameters

`call` UnityAction

## Target(InputTarget)

```
public bool Target(InputTarget newTarget)
```

Parameters

`newTarget` InputTarget

Returns

bool ↗

## Untarget()

```
public InputTarget Untarget()
```

Returns

[InputTarget](#)

# Class InputTarget

Namespace: [PAC.Input](#)

```
public class InputTarget : MonoBehaviour
```

## Inheritance

[object](#) ↗ ← InputTarget

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### allowHoldingKeySpam

```
public bool allowHoldingKeySpam
```

Field Value

[bool](#) ↗

### allowLeftClick

```
public bool allowLeftClick
```

Field Value

[bool](#) ↗

### allowMiddleClick

```
public bool allowMiddleClick
```

Field Value

[bool](#) ↴

## allowRightClick

```
public bool allowRightClick
```

Field Value

[bool](#) ↴

## allowScroll

```
public bool allowScroll
```

Field Value

[bool](#) ↴

## cursorHover

```
public CursorState cursorHover
```

Field Value

[CursorState](#)

## cursorPress

```
public CursorState cursorPress
```

Field Value

[CursorState](#)

## cursorSelected

`public CursorState cursorSelected`

Field Value

[CursorState](#)

## disableMouseWhenSelected

`public bool disableMouseWhenSelected`

Field Value

[bool](#)

## isHoverTrigger

`public bool isHoverTrigger`

Field Value

[bool](#)

## keyboardInputEnabled

`public bool keyboardInputEnabled`

Field Value

[bool](#)

## mouseDeselectMode

```
public MouseTargetDeselectMode mouseDeselectMode
```

### Field Value

[MouseTargetDeselectMode](#)

## mouseInputEnabled

```
public bool mouseInputEnabled
```

### Field Value

[bool](#)

## receiveAlreadyHeldKeys

```
public bool receiveAlreadyHeldKeys
```

### Field Value

[bool](#)

## uiElement

```
public UIElement uiElement
```

### Field Value

[UIElement](#)

## viewport

```
public UIPortage viewport
```

Field Value

[UIPortage](#)

## Properties

collider

```
public Collider2D collider { get; }
```

Property Value

Collider2D

cursorScroll

```
public CursorState cursorScroll { get; set; }
```

Property Value

[CursorState](#)

keyboardTarget

```
public KeyboardTarget keyboardTarget { get; }
```

Property Value

[KeyboardTarget](#)

mouseTarget

```
public MouseTarget mouseTarget { get; }
```

Property Value

[MouseTarget](#)

## targetName

```
public string targetName { get; }
```

Property Value

[string](#)

## targetTag

```
public int targetTag { get; }
```

Property Value

[int](#)

## Methods

### GetUntargeted()

```
public void GetUntargeted()
```

### SubscribeToTarget(UnityAction)

```
public void SubscribeToTarget(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToUntarget(UnityAction)

```
public void SubscribeToUntarget(UnityAction call)
```

Parameters

`call` UnityAction

## Target()

```
public void Target()
```

## Untarget()

```
public void Untarget()
```

# Class Keyboard

Namespace: [PAC.Input](#)

```
public class Keyboard : MonoBehaviour
```

## Inheritance

[object](#) ← Keyboard

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### canInteract

```
public bool canInteract
```

#### Field Value

[bool](#)

# Class KeyboardTarget

Namespace: [PAC.Input](#)

```
public class KeyboardTarget
```

## Inheritance

[object](#) ← KeyboardTarget

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### KeyboardTarget()

```
public KeyboardTarget()
```

## Fields

### allowHoldingKeySpam

```
public bool allowHoldingKeySpam
```

## Field Value

[bool](#)

## receiveAlreadyHeldKeys

```
public bool receiveAlreadyHeldKeys
```

Field Value

[bool](#)

## Properties

### inputThisFrame

```
public bool inputThisFrame { get; }
```

Property Value

[bool](#)

### keysHeld

The keys that are currently held down, in order of when they were pressed (most recent first).

```
public CustomKeyCode[] keysHeld { get; }
```

Property Value

[CustomKeyCode\[\]](#)

### keysPressed

```
public CustomKeyCode[] keysPressed { get; }
```

Property Value

## Methods

### IsHeld(CustomKeyCode)

Returns true if all the given key (and potentially some other keys) is held (and is a key detectable by KeyboardTarget).

```
public bool IsHeld(CustomKeyCode key)
```

Parameters

key [CustomKeyCode](#)

Returns

[bool](#)

### IsHeld(params CustomKeyCode[])

Returns true if all the given keys (and potentially some other keys) are held (and are keys detectable by KeyboardTarget).

```
public bool IsHeld(params CustomKeyCode[] keys)
```

Parameters

keys [CustomKeyCode\[\]](#)

Returns

[bool](#)

### IsHeld(KeyboardShortcut)

Returns true if all the given key (and potentially some other keys) is held (and is a key detectable by KeyboardTarget).

```
public bool IsHeld(KeyboardShortcut shortcut)
```

Parameters

shortcut [KeyboardShortcut](#)

Returns

[bool](#)

## IsHeldExactly(params CustomKeyCode[])

Returns true if all and only the given keys are held (and are keys detectable by KeyboardTarget).

```
public bool IsHeldExactly(params CustomKeyCode[] keys)
```

Parameters

keys [CustomKeyCode](#)[]

Returns

[bool](#)

## IsHeldExactly(KeyboardShortcut)

Returns true if all and only the given keys are held (and are keys detectable by KeyboardTarget).

```
public bool IsHeldExactly(KeyboardShortcut shortcut)
```

Parameters

shortcut [KeyboardShortcut](#)

Returns

[bool](#)

## IsPressed(CustomKeyCode)

Returns true if all the given key (and potentially some other keys) has been pressed this frame (and is a key detectable by KeyboardTarget).

```
public bool IsPressed(CustomKeyCode key)
```

Parameters

key [CustomKeyCode](#)

Returns

[bool](#)

## IsPressed(params CustomKeyCode[])

Returns true if all the given keys (and potentially some other keys) have been pressed this frame (and are keys detectable by KeyboardTarget).

```
public bool IsPressed(params CustomKeyCode[] keys)
```

Parameters

keys [CustomKeyCode\[\]](#)

Returns

[bool](#)

## IsPressed(KeyboardShortcut)

Returns true if all the given keys (and potentially some other keys) have been pressed this frame (and are keys detectable by KeyboardTarget).

```
public bool IsPressed(KeyboardShortcut shortcut)
```

Parameters

`shortcut` [KeyboardShortcut](#)

Returns

`bool`

## KeyDown(CustomKeyCode)

Simulates the key being pressed, if it is a key detectable by KeyboardTarget.

```
public void KeyDown(CustomKeyCode key)
```

Parameters

`key` [CustomKeyCode](#)

## KeyDownNoSpamReset(CustomKeyCode)

Simulates the key being pressed, if it is a key detectable by KeyboardTarget, without restting the timer until key spamming occurs.

```
public void KeyDownNoSpamReset(CustomKeyCode key)
```

Parameters

`key` [CustomKeyCode](#)

## KeyUp(CustomKeyCode)

Simulates the key being unpressed.

```
public void KeyUp(CustomKeyCode key)
```

Parameters

key [CustomKeyCode](#)

## KeysDown(params CustomKeyCode[])

Simulates the keys being pressed, if they are keys detectable by KeyboardTarget.

```
public void KeysDown(params CustomKeyCode[] keys)
```

Parameters

keys [CustomKeyCode](#)[]

## KeysDownNoSpamReset(params CustomKeyCode[])

Simulates the keys being pressed, if they are keys detectable by KeyboardTarget, without restting the timer until key spamming occurs.

```
public void KeysDownNoSpamReset(params CustomKeyCode[] keys)
```

Parameters

keys [CustomKeyCode](#)[]

## KeysUp(params CustomKeyCode[])

Simulates the keys being unpressed.

```
public void KeysUp(params CustomKeyCode[] keys)
```

Parameters

keys [CustomKeyCode\[\]](#)

## ManualUpdate()

Only to be called in InputTarget's Update() method.

```
public void ManualUpdate()
```

## OneIsHeld(IEnumerable<KeyboardShortcut>)

Returns true if all the given keys (and potentially some other keys) are held (and are keys detectable by KeyboardTarget) for one of the given keyboard shortcuts.

```
public bool OneIsHeld(IEnumerable<KeyboardShortcut> shortcuts)
```

Parameters

shortcuts [IEnumerable<KeyboardShortcut>](#)

Returns

[bool](#)

## OneIsHeldExactly(IEnumerable<KeyboardShortcut>)

Returns true if all and only the given keys are held (and are keys detectable by KeyboardTarget) for one of the given keyboard shortcuts.

```
public bool OneIsHeldExactly(IEnumerable<KeyboardShortcut> shortcuts)
```

Parameters

shortcuts [IEnumerable<KeyboardShortcut>](#)

Returns

bool ↴

## OnIsPressed(IEnumerable<KeyboardShortcut>)

Returns true if all the given keys (and potentially some other keys) have been pressed this frame (and are keys detectable by KeyboardTarget) for one of the given keyboard shortcuts.

```
public bool OneIsPressed(IEnumerable<KeyboardShortcut> shortcuts)
```

Parameters

`shortcuts` IEnumerable ↴<KeyboardShortcut>

Returns

bool ↴

## SubscribeToOnInput(UnityAction)

```
public void SubscribeToOnInput(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToOnKeyDown(UnityAction<CustomKeyCode> )

```
public void SubscribeToOnKeyDown(UnityAction<CustomKeyCode> call)
```

Parameters

`call` UnityAction<CustomKeyCode>

## SubscribeToOnKeyUp(UnityAction<CustomKeyCode>)

```
public void SubscribeToOnKeyUp(UnityAction<CustomKeyCode> call)
```

### Parameters

**call** UnityAction<[CustomKeyCode](#)>

## SubscribeToUntarget(UnityAction)

```
public void SubscribeToUntarget(UnityAction call)
```

### Parameters

**call** UnityAction

## Untarget()

```
public void Untarget()
```

# Class Mouse

Namespace: [PAC.Input](#)

```
public class Mouse : MonoBehaviour
```

## Inheritance

[object](#) ← Mouse

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### canInteract

```
public bool canInteract
```

Field Value

[bool](#)

### lockAllUIInteractions

```
public bool lockAllUIInteractions
```

Field Value

[bool](#)

## Properties

### click

```
public bool click { get; }
```

Property Value

[bool](#) ↗

## cursorState

```
public CursorState cursorState { get; }
```

Property Value

[CursorState](#)

## hasHoverTrigger

```
public bool hasHoverTrigger { get; }
```

Property Value

[bool](#) ↗

## held

```
public bool held { get; }
```

Property Value

[bool](#) ↗

## hoverTarget

```
public InputTarget hoverTarget { get; }
```

Property Value

[InputTarget](#)

## leftClick

```
public bool leftClick { get; }
```

Property Value

[bool](#)

## leftHold

```
public bool leftHold { get; }
```

Property Value

[bool](#)

## leftUnclick

```
public bool leftUnclick { get; }
```

Property Value

[bool](#)

## middleClick

```
public bool middleClick { get; }
```

Property Value

[bool](#) ↗

## middleHold

```
public bool middleHold { get; }
```

Property Value

[bool](#) ↗

## middleUnclick

```
public bool middleUnclick { get; }
```

Property Value

[bool](#) ↗

## nothingClick

```
public bool nothingClick { get; }
```

Property Value

[bool](#) ↗

## nothingHeld

```
public bool nothingHeld { get; }
```

Property Value

[bool](#) ↗

## numButtonsHeld

```
public int numButtonsHeld { get; }
```

Property Value

[int](#) ↗

## rightClick

```
public bool rightClick { get; }
```

Property Value

[bool](#) ↗

## rightHold

```
public bool rightHold { get; }
```

Property Value

[bool](#) ↗

## rightUnclick

```
public bool rightUnclick { get; }
```

Property Value

[bool](#)

## scrollCursorDisplayTime

```
public float scrollCursorDisplayTime { get; }
```

Property Value

[float](#)

## scrollDelta

```
public float scrollDelta { get; }
```

Property Value

[float](#)

## unclick

```
public bool unclick { get; }
```

Property Value

[bool](#)

## worldPos

```
public Vector2 worldPos { get; }
```

Property Value

Vector2

## Methods

### IsClicked(MouseButton)

```
public bool IsClicked(MouseButton mouseButton)
```

Parameters

mouseButton [MouseButton](#)

Returns

[bool](#)

### IsHeld(MouseButton)

```
public bool IsHeld(MouseButton mouseButton)
```

Parameters

mouseButton [MouseButton](#)

Returns

[bool](#)

### IsUnclicked(MouseButton)

```
public bool IsUnclicked(MouseButton mouseButton)
```

Parameters

mouseButton [MouseButton](#)

Returns

[bool](#)

## SetCursorSprite(CursorState)

```
public void SetCursorSprite(CursorState cursorState)
```

Parameters

cursorState [CursorState](#)

## SetCursorState(CursorState)

```
public void SetCursorState(CursorState cursorState)
```

Parameters

cursorState [CursorState](#)

## SubscribeToClick(UnityAction)

```
public void SubscribeToClick(UnityAction call)
```

Parameters

call [UnityAction](#)

## SubscribeToLeftClick(UnityAction)

```
public void SubscribeToLeftClick(UnityAction call)
```

### Parameters

`call` UnityAction

## SubscribeToLeftUnclick(UnityAction)

```
public void SubscribeToLeftUnclick(UnityAction call)
```

### Parameters

`call` UnityAction

## SubscribeToMiddleClick(UnityAction)

```
public void SubscribeToMiddleClick(UnityAction call)
```

### Parameters

`call` UnityAction

## SubscribeToMiddleUnclick(UnityAction)

```
public void SubscribeToMiddleUnclick(UnityAction call)
```

### Parameters

`call` UnityAction

## SubscribeToRightClick(UnityAction)

```
public void SubscribeToRightClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToRightUnclick(UnityAction)

```
public void SubscribeToRightUnclick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToScroll(UnityAction)

```
public void SubscribeToScroll(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToUnclick(UnityAction)

```
public void SubscribeToUnclick(UnityAction call)
```

Parameters

`call` UnityAction

# Enum MouseButton

Namespace: [PAC.Input](#)

```
public enum MouseButton
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Left = 0

Middle = 2

Right = 1

# Class MouseTarget

Namespace: [PAC.Input](#)

```
public class MouseTarget
```

## Inheritance

[object](#) ← MouseTarget

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### allowLeftClick

```
public bool allowLeftClick
```

Field Value

[bool](#)

### allowMiddleClick

```
public bool allowMiddleClick
```

Field Value

[bool](#)

## allowRightClick

```
public bool allowRightClick
```

Field Value

[bool](#) ↗

## allowScroll

```
public bool allowScroll
```

Field Value

[bool](#) ↗

## buttonTargetedWith

```
public MouseButton buttonTargetedWith
```

Field Value

[MouseButton](#)

## cursorHover

```
public CursorState cursorHover
```

Field Value

[CursorState](#)

## cursorPress

```
public CursorState cursorPress
```

Field Value

[CursorState](#)

## cursorScroll

```
public CursorState cursorScroll
```

Field Value

[CursorState](#)

## cursorSelected

```
public CursorState cursorSelected
```

Field Value

[CursorState](#)

## deselectMode

```
public MouseTargetDeselectMode deselectMode
```

Field Value

[MouseTargetDeselectMode](#)

## isHoverTrigger

```
public bool isHoverTrigger
```

Field Value

[bool](#) ↗

## Properties

### selected

```
public bool selected { get; }
```

Property Value

[bool](#) ↗

### state

```
public MouseTargetState state { get; }
```

Property Value

[MouseTargetState](#)

### timeHovered

```
public float timeHovered { get; }
```

Property Value

[float](#) ↗

## Methods

## Deselect()

```
public void Deselect()
```

## Hover()

```
public void Hover()
```

## Idle()

```
public void Idle()
```

## Press()

```
public void Press()
```

## Scroll()

```
public void Scroll()
```

## Select()

```
public void Select()
```

## SubscribeToClick(UnityAction)

```
public void SubscribeToClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToDeselect(UnityAction)

```
public void SubscribeToDeselect(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToHover(UnityAction)

```
public void SubscribeToHover(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToIdle(UnityAction)

```
public void SubscribeToIdle(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToLeftClick(UnityAction)

```
public void SubscribeToLeftClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToLeftUnclick(UnityAction)

```
public void SubscribeToLeftUnclick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToMiddleClick(UnityAction)

```
public void SubscribeToMiddleClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToMiddleUnclick(UnityAction)

```
public void SubscribeToMiddleUnclick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToPress(UnityAction)

```
public void SubscribeToPress(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToRightClick(UnityAction)

```
public void SubscribeToRightClick(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToRightUnclick(UnityAction)

```
public void SubscribeToRightUnclick(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToScroll(UnityAction)

```
public void SubscribeToScroll(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToSelect(UnityAction)

```
public void SubscribeToSelect(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToStateChange(UnityAction)

```
public void SubscribeToStateChange(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToUnclick(UnityAction)

```
public void SubscribeToUnclick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToUntarget(UnityAction)

```
public void SubscribeToUntarget(UnityAction call)
```

Parameters

`call` UnityAction

## Untarget()

```
public void Untarget()
```

# Enum MouseTargetDeselectMode

Namespace: [PAC.Input](#)

```
public enum MouseTargetDeselectMode
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

clickAgain = 1

Manual = 100

Unclick = 0

# Enum MouseTargetState

Namespace: [PAC.Input](#)

```
public enum MouseTargetState
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Hover = 1

Idle = 0

Pressed = 2

# Class Tooltip

Namespace: [PAC.Input](#)

```
public class Tooltip : MonoBehaviour
```

## Inheritance

[object](#) ← Tooltip

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### hoverTimeUntilTooltip

```
public float hoverTimeUntilTooltip
```

Field Value

[float](#)

### padding

```
public Vector2 padding
```

Field Value

Vector2

## Properties

### text

```
public string text { get; set; }
```

Property Value

[string](#)

## tooltipVisible

```
public bool tooltipVisible { get; }
```

Property Value

[bool](#)

## Methods

### HideTooltip()

```
public void HideTooltip()
```

### ShowTooltip(Vector2)

```
public void ShowTooltip(Vector2 worldCoords)
```

Parameters

**worldCoords** Vector2

# Namespace PAC.Interfaces

## Interfaces

### [IReadOnlyContains<T>](#)

Adds [Contains\(T\)](#) to [IReadOnlyCollection<T>](#).

### [ISetComparable<T>](#)

Defines how to determine whether two sets are equal / one is a subset of the other.

# Interface IReadOnlyContains<T>

Namespace: [PAC.Interfaces](#)

Adds [Contains\(T\)](#) to [IReadOnlyCollection<T>](#).

```
public interface IReadOnlyContains<T> : IReadOnlyCollection<T>,  
IEnumerable<T>, IEnumerable
```

## Type Parameters

T

### Inherited Members

[IReadOnlyCollection<T>.Count](#) , [IEnumerable<T>.GetEnumerator\(\)](#)

### Extension Methods

```
IReadOnlyContainsExtensions.ContainsAll<T>(IReadOnlyContains<T>, IEnumerable<T>),  
IReadOnlyContainsExtensions.ContainsAll<T>(IReadOnlyContains<T>, params T[]),  
IReadOnlyContainsExtensions.ContainsAny<T>(IReadOnlyContains<T>, IEnumerable<T>),  
IReadOnlyContainsExtensions.ContainsAny<T>(IReadOnlyContains<T>, params T[]),  
IReadOnlyContainsExtensions.ContainsNone<T>(IReadOnlyContains<T>,  
IEnumerable<T>),  
IReadOnlyContainsExtensions.ContainsNone<T>(IReadOnlyContains<T>, params T[]),  
IReadOnlyContainsExtensions.ContainsNotAll<T>(IReadOnlyContains<T>,  
IEnumerable<T>),  
IReadOnlyContainsExtensions.ContainsNotAll<T>(IReadOnlyContains<T>, params T[]),  
ISetComparableExtensions.IsProperSubsetOf<T1, T2>(T1, T2),  
ISetComparableExtensions.IsProperSupersetOf<T1, T2>(T1, T2),  
IEnumerableExtensions.AreAllDistinct<T>(IEnumerable<T>),  
IEnumerableExtensions.ArgMax<TElement, TCompare>(IEnumerable<TElement>,  
Func<TElement, TCompare>),  
IEnumerableExtensions.ArgMin<TElement, TCompare>(IEnumerable<TElement>,  
Func<TElement, TCompare>),  
IEnumerableExtensions.Concat<T>(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>,  
params IEnumerable<T>[]),  
IEnumerableExtensions.ContainsAll<T>(IEnumerable<T>, IEnumerable<T>),  
IEnumerableExtensions.ContainsAll<T>(IEnumerable<T>, params T[]),  
IEnumerableExtensions.ContainsAny<T>(IEnumerable<T>, IEnumerable<T>),  
IEnumerableExtensions.ContainsAny<T>(IEnumerable<T>, params T[]),
```

[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Methods

### Contains(T)

```
bool Contains(T item)
```

## Parameters

item T

Returns

bool ↗

# Interface ISetComparable<T>

Namespace: [PAC.Interfaces](#)

Defines how to determine whether two sets are equal / one is a subset of the other.

```
public interface ISetComparable<T>
```

## Type Parameters

T

The type to define set comparison against.

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Remarks

If **T1** implements **ISetComparable<T2>** and **T2** implements **ISetComparable<T1>**, it is up to implementers to ensure the logical symmetries of the methods. See the documentation of each method for more detail on these symmetries.

## Methods

### IsSubsetOf(T)

Whether **this** as a set is a subset of **other** as a set.

```
bool IsSubsetOf(T other)
```

## Parameters

**other** T

## Returns

[bool](#)

## Remarks

Recall that '[this](#)' is subset of [other](#)' means that every element of [this](#) is an element of [other](#), ignoring order and duplicate elements.

If [T1](#) implements [ISetComparable<T2>](#) and [T2](#) implements [ISetComparable<T1>](#), then [T1.IsSubsetOf\(T2\)](#) should be logically equivalent to [T2.IsSupersetOf\(T1\)](#).

## See Also

[IsSupersetOf\(T\)](#)

## IsSupersetOf(T)

Whether [this](#) as a set is a superset of [other](#) as a set.

```
bool IsSupersetOf(T other)
```

## Parameters

[other](#) T

## Returns

[bool](#)

## Remarks

Recall that '[this](#)' is superset of [other](#)' means that every element of [other](#) is an element of [this](#), ignoring order and duplicate elements.

If [T1](#) implements [ISetComparable<T2>](#) and [T2](#) implements [ISetComparable<T1>](#), then [T1.IsSupersetOf\(T2\)](#) should be logically equivalent to [T2.IsSubsetOf\(T1\)](#).

## See Also

[IsSubsetOf\(T\)](#)

## SetEquals(T)

Whether [this](#) as a set is equal to [other](#) as a set.

```
bool SetEquals(T other)
```

## Parameters

other T

## Returns

[bool](#)

## Remarks

Recall that set equality means they have precisely the same elements, ignoring order and duplicate elements.

If `T1` implements `ISetComparable<T2>` and `T2` implements `ISetComparable<T1>`, then `T1.SetEquals(T2)` should be logically equivalent to `T2.SetEquals(T1)`.

# Namespace PAC.Json

## Classes

### [JsonConversion](#)

#### [JsonConversion.JsonConverterSet](#)

Represents a set of JsonConverter objects, making sure there's at most one custom converter for each type. For example, a class MyClass cannot have two JSON converters defined for it in the list.

NOTE: Accessing converters is O(1).

#### [JsonConversion.JsonConverter<T,JsonDataType>](#)

An interface that custom JSON converters must implement.

NOTE: Has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

#### [JsonData.Bool](#)

Represents a bool in JSON data.

#### [JsonData.Float](#)

Represents a float in JSON data.

#### [JsonData.Int](#)

Represents an int in JSON data.

#### [JsonData.List](#)

Represents a non-null list/array in JSON data.

#### [JsonData.Null](#)

Represents a null value in JSON data.

#### [JsonData.Object](#)

Represents a non-null object in JSON data.

#### [JsonData.String](#)

Represents a non-null string in JSON data.

## Interfaces

### [JsonData](#)

Represents a piece of JSON data.



# Class JsonConversion

Namespace: [PAC.Json](#)

```
public static class JsonConversion
```

## Inheritance

[object](#) ← JsonConversion

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### FromJson<T>(JsonData, JsonConverterSet, bool)

Attempts to convert the JSON data into a C# object of the given type.

```
public static T FromJson<T>(JsonData jsonData, JsonConversion.JsonConverterSet  
customConverters, bool allowUndefinedConversions = false)
```

#### Parameters

**jsonData** [JsonData](#)

The JSON data to convert.

**customConverters** [JsonConversion.JsonConverterSet](#)

A collection of JsonConverter objects that defined custom conversions for certain data types.

**allowUndefinedConversions** [bool](#)

If false, an exception will be thrown if the code encounters a type that can not be converted using conversions for primitive JSON types and/or custom converters.

Returns

T

Type Parameters

T

The type to convert the JSON into.

**FromJson<T>(JsonData, JsonConverterSet, bool, HashSet<JsonData>)**

Attempts to convert the JSON data into a C# object of the given type.

```
public static T FromJson<T>(JsonData jsonData, JsonConversion.JsonConverterSet  
customConverters, bool allowUndefinedConversions, HashSet<JsonData>  
dataAlreadyTryingToConvert)
```

Parameters

**jsonData** [JsonData](#)

The JSON data to convert.

**customConverters** [JsonConversion.JsonConverterSet](#)

A collection of JsonConverter objects that defined custom conversions for certain data types.

**allowUndefinedConversions** [bool](#)

If false, an exception will be thrown if the code encounters a type that can not be converted using conversions for primitive JSON types and/or custom converters.

**dataAlreadyTryingToConvert** [HashSet](#)<JsonData>

The JSON data objects trying to be converted in the current function call stack. Used to prevent infinite loops when converting objects with circular references.

Returns

T

## Type Parameters

T

The type to convert the JSON into.

## FromJson<T>(JsonData, bool)

Attempts to convert the JSON data into a C# object of the given type.

```
public static T FromJson<T>(JsonData jsonData, bool allowUndefinedConversions  
= false)
```

## Parameters

jsonData [JsonData](#)

The JSON data to convert.

allowUndefinedConversions [bool](#)

If false, an exception will be thrown if the code encounters a type that can not be converted using conversions for primitive JSON types and/or custom converters.

## Returns

T

## Type Parameters

T

The type to convert the JSON into.

## ToJson(object, JsonConverterSet, bool)

Attempts to convert the C# object into JSON data.

NOTE: When converting an enum value which has more than one name, one of those names will be chosen, but not necessarily the one originally used to assign that value. E.g. for an enum with values Value1 = 0, Value2 = 0, Value3 = 1, converting Value1 into JSON may end up as "Value2".

```
public static JsonData ToJson(object obj, JsonConversion.JsonConverterSet  
customConverters, bool allowUndefinedConversions = false)
```

## Parameters

### [obj object](#)

The object to convert into JSON.

### [customConverters JsonConversion.JsonConverterSet](#)

A collection of JsonConverter objects that defined custom conversions for certain data types.

### [allowUndefinedConversions bool](#)

If false, an exception will be thrown if the code encounters a type that can not be converted using conversions for primitive JSON types and/or custom converters.

## Returns

### [JsonData](#)

## ToJson(object, bool)

Attempts to convert the C# object into JSON data.

NOTE: When converting an enum value which has more than one name, one of those names will be chosen, but not necessarily the one originally used to assign that value. E.g. for an enum with values Value1 = 0, Value2 = 0, Value3 = 1, converting Value1 into JSON may end up as "Value2".

```
public static JsonData ToJson(object obj, bool allowUndefinedConversions = false)
```

## Parameters

`obj` [object](#)

The object to convert into JSON.

`allowUndefinedConversions` [bool](#)

If false, an exception will be thrown if the code encounters a type that can not be converted using conversions for primitive JSON types and/or custom converters.

Returns

[JsonData](#)

# Class JsonConversion.JsonConverterSet

Namespace: [PAC.Json](#)

Represents a set of JsonConverter objects, making sure there's at most one custom converter for each type. For example, a class MyClass cannot have two JSON converters defined for it in the list.

NOTE: Accessing converters is O(1).

```
public class JsonConversion.JsonConverterSet : IEnumerable
```

## Inheritance

[object](#) ← JsonConversion.JsonConverterSet

## Implements

[IEnumerable](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### JsonConverterSet()

Creates an empty JsonConverterSet.

```
public JsonConverterSet()
```

### JsonConverterSet(IEnumerable)

```
public JsonConverterSet(IEnumerable converters)
```

## Parameters

converters [IEnumerable](#)

## JsonConverterSet(params object[])

Creates a JsonConverterSet with the given converters. Throws an error if an object is not a type that implements JsonConverter, or if the set already contains a converter for the type a converter converts.

```
public JsonConverterSet(params object[] converters)
```

## Parameters

converters [object](#)[]

# Methods

## Add(object)

Adds the converter to the set. Throws an error if it is not a type that implements JsonConverter, or if the set already contains a converter for the type the converter converts.

```
public void Add(object converter)
```

## Parameters

converter [object](#)

## ContainsConverterFor(Type)

Returns whether the set contains a converter that converts the given type.

```
public bool ContainsConverterFor(Type converterType)
```

## Parameters

converterType [Type](#)

## Returns

[bool](#)

true if the set has a converter for the given type.

## GetConverterFor(Type)

If the set contains a converter that converts the given type, this will return it.

```
public object GetConverterFor(Type converterType)
```

## Parameters

converterType [Type](#)

## Returns

[object](#)

The converter for the given type, if there is one, otherwise null.

## GetEnumerator()

Returns an enumerator that iterates through a collection.

```
public IEnumerator GetEnumerator()
```

## Returns

[IEnumerator](#)

An [IEnumerator](#) object that can be used to iterate through the collection.

## RemoveConverterFor(Type)

If the set contains a converter that converts the given type, this will remove it.

```
public bool RemoveConverterFor(Type converterType)
```

Parameters

converterType [Type](#)

Returns

[bool](#)

true if a converter was removed.

# Class JsonConversion.JsonConverter<T, JsonDataType>

Namespace: [PAC.Json](#)

An interface that custom JSON converters must implement.

NOTE: Has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public abstract class JsonConversion.JsonConverter<T, JsonDataType> where  
JsonDataType : JsonData
```

## Type Parameters

T

The type the converter will convert to/from JSON.

JsonDataType

The type of JSON data the converter will convert to/from.

## Inheritance

[object](#) ↗ ← JsonConversion.JsonConverter<T, JsonDataType>

## Derived

[AnimationKeyFrame.JsonConverter](#), [BlendMode.JsonConverter](#),  
[SemanticVersion.JsonConverter](#), [JsonConverters.ColorJsonConverter](#),  
[JsonConverters.Texture2DJsonConverter](#), [JsonConverters.Vector2JsonConverter](#),  
[JsonConverters.Vector3JsonConverter](#), [File.JsonConverter](#), [CustomKeyCode.JsonConverter](#),  
[KeyboardShortcut.JsonConverter](#), [Layer.JsonConverter](#), [NormalLayer.JsonConverter](#),  
[TileLayer.JsonConverter](#)

## Inherited Members

[object.Equals\(object\)](#) ↗ , [object.Equals\(object, object\)](#) ↗ , [object.GetHashCode\(\)](#) ↗ ,  
[object.GetType\(\)](#) ↗ , [object.MemberwiseClone\(\)](#) ↗ , [object.ReferenceEquals\(object, object\)](#) ↗ ,  
[object.ToString\(\)](#) ↗

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### FromJson(JsonData)

Attempts to convert the JSON data into a C# object of the given type. Ensures the JSON data is the correct type before calling the FromJson() overload for the specific type of JSON data.

```
public T FromJson(JsonData jsonData)
```

#### Parameters

jsonData [JsonData](#)

#### Returns

T

### FromJson(JsonDataType)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public abstract T FromJson(JsonDataType jsonData)
```

#### Parameters

jsonData [JsonDataType](#)

#### Returns

T

## ToJson(T)

Attempts to convert the C# object into JSON data.

```
public abstract JsonDataType ToJson(T obj)
```

Parameters

**obj** T

Returns

JsonDataType

# Interface JsonData

Namespace: [PAC.Json](#)

Represents a piece of JSON data.

```
public interface JsonData
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#),  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### HaveSameData(JsonData, JsonData, float)

Returns whether the two pieces of JSON data are equal by value rather than whether they point to the same JsonData object.

```
public static bool HaveSameData(JsonData jsonData1, JsonData jsonData2, float  
floatTolerance = 0)
```

#### Parameters

jsonData1 [JsonData](#)

jsonData2 [JsonData](#)

floatTolerance [float](#)

A tolerance (inclusive) to allow close floats to be considered equal.

#### Returns

[bool](#)

### Parse(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as JsonData. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData Parse(TextReader reader, bool mustReadAll)
```

## Parameters

reader [TextReader](#)

mustReadAll [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

## Returns

[JsonData](#)

## Parse(string)

Attempts to parse the string as JsonData.

```
public static JsonData Parse(string str)
```

## Parameters

str [string](#)

## Returns

[JsonData](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as JsonData. If successful, the index will be moved to the last character of the data.

```
public static JsonData Parse(string str, ref int index)
```

## Parameters

`str` [string](#)

`index` [int](#)

## Returns

[JsonData](#)

## ParseNumber(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as a JsonData.Int or JsonData.Float. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData ParseNumber(TextReader reader, bool mustReadAll)
```

## Parameters

`reader` [TextReader](#)

`mustReadAll` [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

## Returns

[JsonData](#)

## ParseNumber(string)

Attempts to parse the string into a JsonData.Int or JsonData.Float.

```
public static JsonData ParseNumber(string str)
```

## Parameters

`str` [string](#)

Returns

[JsonData](#)

## ParseNumber(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as a `JsonData.Int` or `JsonData.Float`. If successful, the index will be moved to the last digit of the number.

```
public static JsonData ParseNumber(string str, ref int index)
```

Parameters

`str` [string](#)

`index` [int](#)

Returns

[JsonData](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
string ToJsonObject(bool pretty)
```

Parameters

`pretty` [bool](#)

If true, puts the string into pretty form - e.g. with indentations.

Returns

[string ↗](#)

# Class JsonData.Bool

Namespace: [PAC.Json](#)

Represents a bool in JSON data.

```
public class JsonData.Bool : JsonData
```

## Inheritance

[object](#) ← JsonData.Bool

## Implements

[JsonData](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### Bool(bool)

```
public Bool(bool value)
```

## Parameters

value [bool](#)

## Properties

### value

```
public bool value { get; set; }
```

Property Value

[bool](#)

## Methods

### Parse(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as a JsonData.Bool. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData.Bool Parse(TextReader reader, bool mustReadAll)
```

Parameters

reader [TextReader](#)

mustReadAll [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

Returns

[JsonData.Bool](#)

### Parse(string)

Attempts to parse the string into a JsonData.Bool.

```
public static JsonData.Bool Parse(string str)
```

Parameters

`str` [string](#)

Returns

[JsonData.Bool](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as `JsonData.Bool`. If successful, the index will be moved to the last character of the null.

```
public static JsonData.Bool Parse(string str, ref int index)
```

Parameters

`str` [string](#)

`index` [int](#)

Returns

[JsonData.Bool](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
public string ToJsonObject(bool pretty)
```

Parameters

`pretty` [bool](#)

If true, puts the string into pretty form - e.g. with indentations.

Returns

[string](#)

# Operators

## implicit operator bool(Bool)

```
public static implicit operator bool(JsonData.Bool jsonBool)
```

### Parameters

jsonBool [JsonData.Bool](#)

### Returns

[bool](#)

## implicit operator Bool(bool)

```
public static implicit operator JsonData.Bool(bool value)
```

### Parameters

value [bool](#)

### Returns

[JsonData.Bool](#)

# Class jsonData.Float

Namespace: [PAC.Json](#)

Represents a float in JSON data.

```
public class jsonData.Float : jsonData
```

## Inheritance

[object](#) ← jsonData.Float

## Implements

[jsonData](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### Float(float)

```
public Float(float value)
```

## Parameters

value [float](#)

## Properties

### value

```
public float value { get; set; }
```

Property Value

[float](#)

## Methods

### Parse(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as a JsonData.Float. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData.Float Parse(TextReader reader, bool mustReadAll)
```

Parameters

reader [TextReader](#)

mustReadAll [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

Returns

[JsonData.Float](#)

### Parse(string)

Attempts to parse the string into a JsonData.Float.

```
public static JsonData.Float Parse(string str)
```

Parameters

`str` [string](#)

Returns

[JsonData.Float](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as a `JsonData.Float`. If successful, the index will be moved to the last digit of the float.

```
public static JsonData.Float Parse(string str, ref int index)
```

Parameters

`str` [string](#)

`index` [int](#)

Returns

[JsonData.Float](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
public string ToJsonObject(bool pretty)
```

Parameters

`pretty` [bool](#)

If true, puts the string into pretty form - e.g. with indentations.

Returns

[string](#)

# Operators

## explicit operator int(Float)

```
public static explicit operator int(JsonData.Float jsonfloat)
```

### Parameters

jsonfloat [JsonData.Float](#)

### Returns

[int](#)

## implicit operator float(Float)

```
public static implicit operator float(JsonData.Float jsonfloat)
```

### Parameters

jsonfloat [JsonData.Float](#)

### Returns

[float](#)

## implicit operator Float(int)

```
public static implicit operator JsonData.Float(int value)
```

### Parameters

value [int](#)

### Returns

## implicit operator Float(float)

```
public static implicit operator JsonData.Float(float value)
```

### Parameters

**value** [float](#)

### Returns

[JsonData](#).[Float](#)

# Class jsonData.Int

Namespace: [PAC.Json](#)

Represents an int in JSON data.

```
public class jsonData.Int : jsonData
```

## Inheritance

[object](#) ← jsonData.Int

## Implements

[jsonData](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### Int(int)

```
public Int(int value)
```

## Parameters

value [int](#)

## Properties

### value

```
public int value { get; set; }
```

Property Value

[int](#)

## Methods

### Parse(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as a JsonData.Int. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData.Int Parse(TextReader reader, bool mustReadAll)
```

Parameters

reader [TextReader](#)

mustReadAll [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

Returns

[JsonData.Int](#)

### Parse(string)

Attempts to parse the string into a JsonData.Int.

```
public static JsonData.Int Parse(string str)
```

Parameters

`str` [string](#)

Returns

[JsonData.Int](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as a `JsonData.Int`. If successful, the index will be moved to the last digit of the int.

```
public static JsonData.Int Parse(string str, ref int index)
```

Parameters

`str` [string](#)

`index` [int](#)

Returns

[JsonData.Int](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
public string ToJsonObject(bool pretty)
```

Parameters

`pretty` [bool](#)

If true, puts the string into pretty form - e.g. with indentations.

Returns

[string](#)

# Operators

## explicit operator Int(float)

```
public static explicit operator JsonData.Int(float value)
```

### Parameters

value [float](#)

### Returns

[JsonData.Int](#)

## implicit operator int(Int)

```
public static implicit operator int(JsonData.Int jsonInt)
```

### Parameters

jsonInt [JsonData.Int](#)

### Returns

[int](#)

## implicit operator float(Int)

```
public static implicit operator float(JsonData.Int jsonInt)
```

### Parameters

jsonInt [JsonData.Int](#)

### Returns

[float](#)

## implicit operator Int(int)

```
public static implicit operator JsonData.Int(int value)
```

### Parameters

**value** [int](#)

### Returns

[JsonData.Int](#)

# Class jsonData.List

Namespace: [PAC.Json](#)

Represents a non-null list/array in JSON data.

```
public class jsonData.List : List<jsonData>, IList<jsonData>, ICollection<jsonData>,  
IReadOnlyList<jsonData>, IReadOnlyCollection<jsonData>, IEnumerable<jsonData>,  
IList, ICollection, IEnumerable, jsonData
```

## Inheritance

[object](#) ← [List](#)<[jsonData](#)> ← jsonData.List

## Implements

[IList](#)<[jsonData](#)>, [ICollection](#)<[jsonData](#)>, [IReadOnlyList](#)<[jsonData](#)>,  
[IReadOnlyCollection](#)<[jsonData](#)>, [IEnumerable](#)<[jsonData](#)>, [IList](#), [ICollection](#),  
[IEnumerable](#), [jsonData](#)

## Inherited Members

[List](#)<[jsonData](#)>.Add([jsonData](#)) , [List](#)<[jsonData](#)>.AddRange([IEnumerable](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.AsReadOnly() ,  
[List](#)<[jsonData](#)>.BinarySearch([int](#), [int](#), [jsonData](#), [IComparer](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.BinarySearch([jsonData](#)) ,  
[List](#)<[jsonData](#)>.BinarySearch([jsonData](#), [IComparer](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.Clear() , [List](#)<[jsonData](#)>.Contains([jsonData](#)) ,  
[List](#)<[jsonData](#)>.ConvertAll<[TOutput](#)>([Converter](#)<[jsonData](#), [TOutput](#)>) ,  
[List](#)<[jsonData](#)>.CopyTo([int](#), [jsonData](#)[], [int](#), [int](#)) , [List](#)<[jsonData](#)>.CopyTo([jsonData](#)[]) ,  
[List](#)<[jsonData](#)>.CopyTo([jsonData](#)[], [int](#)) , [List](#)<[jsonData](#)>.EnsureCapacity([int](#)) ,  
[List](#)<[jsonData](#)>.Exists([Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.Find([Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindAll([Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindIndex([int](#), [int](#), [Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindIndex([int](#), [Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindIndex([Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindLast([Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindLastIndex([int](#), [int](#), [Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindLastIndex([int](#), [Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.FindLastIndex([Predicate](#)<[jsonData](#)>) ,  
[List](#)<[jsonData](#)>.ForEach([Action](#)<[jsonData](#)>) , [List](#)<[jsonData](#)>.GetEnumerator() ,  
[List](#)<[jsonData](#)>.GetRange([int](#), [int](#)) , [List](#)<[jsonData](#)>.IndexOf([jsonData](#)) ,

[List<JsonData>.IndexOf\(JsonData, int\)](#) , [List<JsonData>.IndexOf\(JsonData, int, int\)](#) ,  
[List<JsonData>.Insert\(int, JsonData\)](#) ,  
[List<JsonData>.InsertRange\(int, IEnumerable<JsonData>\)](#) ,  
[List<JsonData>.LastIndexOf\(JsonData\)](#) , [List<JsonData>.LastIndexOf\(JsonData, int\)](#) ,  
[List<JsonData>.LastIndexOf\(JsonData, int, int\)](#) , [List<JsonData>.Remove\(JsonData\)](#) ,  
[List<JsonData>.RemoveAll\(Predicate<JsonData>\)](#) , [List<JsonData>.RemoveAt\(int\)](#) ,  
[List<JsonData>.RemoveRange\(int, int\)](#) , [List<JsonData>.Reverse\(\)](#) ,  
[List<JsonData>.Reverse\(int, int\)](#) , [List<JsonData>.Slice\(int, int\)](#) ,  
[List<JsonData>.Sort\(\)](#) , [List<JsonData>.Sort\(IComparer<JsonData>\)](#) ,  
[List<JsonData>.Sort\(Comparison<JsonData>\)](#) ,  
[List<JsonData>.Sort\(int, int, IComparer<JsonData>\)](#) , [List<JsonData>.ToArray\(\)](#) ,  
[List<JsonData>.TrimExcess\(\)](#) , [List<JsonData>.TrueForAll\(Predicate<JsonData>\)](#) ,  
[List<JsonData>.Capacity](#) , [List<JsonData>.Count](#) , [List<JsonData>.this\[int\]](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[ICollectionExtensions.ContainsAll<T>\(ICollection<T>, params T\[\]\)](#) ,  
[ICollectionExtensions.ContainsAll<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[ICollectionExtensions.ContainsAny<T>\(ICollection<T>, params T\[\]\)](#) ,  
[ICollectionExtensions.ContainsAny<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[ICollectionExtensions.ContainsNone<T>\(ICollection<T>, params T\[\]\)](#) ,  
[ICollectionExtensions.ContainsNone<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[ICollectionExtensions.ContainsNotAll<T>\(ICollection<T>, params T\[\]\)](#) ,  
[ICollectionExtensions.ContainsNotAll<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,

[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#) ,  
[IReadOnlyListExtensions.GetRange<T>\(IReadOnlyList<T>, int, int\)](#) ,  
[IReadOnlyListExtensions.GetRange<T>\(IReadOnlyList<T>, Range\)](#) ,  
[IntRangeExtensions.GetRange<T>\(IReadOnlyList<T>, IntRange\)](#).

## Constructors

### List(params Bool[])

```
public List(params JsonData.Bool[] jsonData)
```

## Parameters

jsonData [Bool\[\]](#)

## List(params Float[])

```
public List(params JsonData.Float[] jsonData)
```

### Parameters

jsonData [Float](#)[]

## List(params Int[])

```
public List(params JsonData.Int[] jsonData)
```

### Parameters

jsonData [Int](#)[]

## List(params List[])

```
public List(params JsonData.List[] jsonData)
```

### Parameters

jsonData [List](#)[]

## List(params Null[])

```
public List(params JsonData.Null[] jsonData)
```

### Parameters

jsonData [Null](#)[]

## List(params Object[])

```
public List(params jsonData.Object[] jsonData)
```

Parameters

jsonData [Object\[\]](#)

## List(params String[])

```
public List(params jsonData.String[] jsonData)
```

Parameters

jsonData [String\[\]](#)

## List(params JsonData[])

```
public List(params jsonData.JsonData[] jsonData)
```

Parameters

jsonData [JsonData\[\]](#)

## List(bool)

```
public List(bool separateLinesInPrettyString = true)
```

Parameters

separateLinesInPrettyString [bool](#)

## List(params bool[])

```
public List(params bool[] jsonData)
```

Parameters

jsonData [bool\[\]](#)

## List(IEnumerable<Bool>)

```
public List(IEnumerable<JsonData.Bool> collection)
```

Parameters

collection [IEnumerable<JsonData.Bool>](#)

## List(IEnumerable<Float>)

```
public List(IEnumerable<JsonData.Float> collection)
```

Parameters

collection [IEnumerable<JsonData.Float>](#)

## List(IEnumerable<Int>)

```
public List(IEnumerable<JsonData.Int> collection)
```

Parameters

collection [IEnumerable<JsonData.Int>](#)

## List(IEnumerable<List>)

```
public List(IEnumerable<JsonData.List> collection)
```

Parameters

collection [IEnumerable](#)<[JsonData](#).[List](#)>

## List(IEnumerable<Null>)

```
public List(IEnumerable<JsonData.Null> collection)
```

Parameters

collection [IEnumerable](#)<[JsonData](#).[Null](#)>

## List(IEnumerable<Object>)

```
public List(IEnumerable<JsonData.Object> collection)
```

Parameters

collection [IEnumerable](#)<[JsonData](#).[Object](#)>

## List(IEnumerable<String>)

```
public List(IEnumerable<JsonData.String> collection)
```

Parameters

collection [IEnumerable](#)<[JsonData](#).[String](#)>

## List(IEnumerable<JsonData>)

```
public List(IEnumerable<JsonData> collection)
```

Parameters

collection [IEnumerable](#)<JsonData>

## List(IEnumerable<bool>)

```
public List(IEnumerable<bool> collection)
```

Parameters

collection [IEnumerable](#)<bool>

## List(IEnumerable<int>)

```
public List(IEnumerable<int> collection)
```

Parameters

collection [IEnumerable](#)<int>

## List(IEnumerable<float>)

```
public List(IEnumerable<float> collection)
```

Parameters

collection [IEnumerable](#)<float>

## List(IEnumerable<string>)

```
public List(IEnumerable<string> collection)
```

Parameters

collection [IEnumerable](#)<[string](#)>

## List(int)

```
public List(int capacity)
```

Parameters

capacity [int](#)

## List(params int[])

```
public List(params int[] jsonData)
```

Parameters

jsonData [int](#)[]

## List(params float[])

```
public List(params float[] jsonData)
```

Parameters

jsonData [float](#)[]

## List(params string[])

```
public List<params string[]> jsonData)
```

Parameters

jsonData [string](#)[]

## Properties

### separateLinesInPrettyString

```
public bool separateLinesInPrettyString { get; set; }
```

Property Value

[bool](#)

## Methods

### Add(Bool)

```
public void Add(JsonData.Bool jsonData)
```

Parameters

jsonData [JsonData.Bool](#)

### Add(Float)

```
public void Add(JsonData.Float jsonData)
```

Parameters

jsonData [JsonData.Float](#)

## Add(Int)

```
public void Add(JsonData.Int jsonData)
```

Parameters

jsonData [JsonData.Int](#)

## Add(List)

```
public void Add(JsonData.List jsonData)
```

Parameters

jsonData [JsonData.List](#)

## Add(Null)

```
public void Add(JsonData.Null jsonData)
```

Parameters

jsonData [JsonData.Null](#)

## Add(Object)

```
public void Add(JsonData.Object jsonData)
```

Parameters

jsonData [JsonData.Object](#)

## Add(String)

```
public void Add(JsonData.String jsonData)
```

Parameters

jsonData [JsonData.String](#)

## Add(bool)

```
public void Add(bool jsonData)
```

Parameters

jsonData [bool](#)

## Add(int)

```
public void Add(int jsonData)
```

Parameters

jsonData [int](#)

## Add(float)

```
public void Add(float jsonData)
```

Parameters

jsonData [float](#)

## Add(string)

```
public void Add(string jsonData)
```

Parameters

jsonData [string](#)

## Parse(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as a JsonData.List. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData.List Parse(TextReader reader, bool mustReadAll)
```

Parameters

reader [TextReader](#)

mustReadAll [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

Returns

[JsonData.List](#)

## Parse(string)

Attempts to parse the string into a JsonData.List.

```
public static JsonData.List Parse(string str)
```

Parameters

str [string](#)

Returns

[JsonData.List](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as a JsonData.List. If successful, the index will be moved to the closing square bracket.

```
public static JsonData.List Parse(string str, ref int index)
```

Parameters

str [string](#)

index [int](#)

Returns

[JsonData.List](#)

## ParseMaybeNull(string)

Attempts to parse the string into a JSON list, with the potential to parse it into JsonData.Null.

```
public static JsonData ParseMaybeNull(string str)
```

Parameters

str [string](#)

Returns

[JsonData](#)

## ParseMaybeNull(string, ref int)

Attempts to parse the string into a JSON list, with the potential to parse it into `JsonData.Null`. If successful, moves the index to the last parsed character.

```
public static JsonData ParseMaybeNull(string str, ref int index)
```

Parameters

`str` [string](#)

`index` [int](#)

Returns

[JsonData](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
public string ToJsonObject(bool pretty)
```

Parameters

`pretty` [bool](#)

If true, puts the string into pretty form - e.g. with indentations.

Returns

[string](#)

# Class jsonData.Null

Namespace: [PAC.Json](#)

Represents a null value in JSON data.

```
public class jsonData.Null : jsonData
```

## Inheritance

[object](#) ← jsonData.Null

## Implements

[jsonData](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### Null()

```
public Null()
```

## Methods

### Parse(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as jsonData.Null. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData.Null Parse(TextReader reader, bool mustReadAll)
```

## Parameters

reader [TextReader](#)

mustReadAll [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

## Returns

[JsonData.Null](#)

## Parse(string)

Attempts to parse the string as JsonData.Null.

```
public static JsonData.Null Parse(string str)
```

## Parameters

str [string](#)

## Returns

[JsonData.Null](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as JsonData.Null. If successful, the index will be moved to the last character of the null.

```
public static JsonData.Null Parse(string str, ref int index)
```

## Parameters

`str` [string](#)

`index` [int](#)

Returns

[JsonData.Null](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
public string ToJsonObject(bool pretty)
```

Parameters

`pretty` [bool](#)

If true, puts the string into pretty form - e.g. with indentations.

Returns

[string](#)

## Operators

### implicit operator List(Null)

```
public static implicit operator JsonData.List(JsonData.Null jsonNull)
```

Parameters

`jsonNull` [JsonData.Null](#)

Returns

[JsonData.List](#)

## implicit operator Object(Null)

```
public static implicit operator jsonData.Object(jsonData.Null jsonNull)
```

Parameters

jsonNull [jsonData.Null](#)

Returns

[jsonData.Object](#)

## implicit operator string(Null)

```
public static implicit operator string(jsonData.Null jsonNull)
```

Parameters

jsonNull [jsonData.Null](#)

Returns

[string](#) ↗

# Class jsonData.Object

Namespace: [PAC.Json](#)

Represents a non-null object in JSON data.

```
public class jsonData.Object : Dictionary<string, jsonData>, IDictionary<string, jsonData>, ICollection<KeyValuePair<string, jsonData>>, IReadOnlyDictionary<string, jsonData>, IReadOnlyCollection<KeyValuePair<string, jsonData>>, IEnumerable<KeyValuePair<string, jsonData>>, IDictionary, ICollection, IEnumerable, IDeserializationCallback, ISerializable, jsonData
```

## Inheritance

[object](#) ← [Dictionary](#)<[string](#), [jsonData](#)> ← [jsonData.Object](#)

## Implements

[IDictionary](#)<[string](#), [jsonData](#)>, [ICollection](#)<[KeyValuePair](#)<[string](#), [jsonData](#)>>,  
[IReadOnlyDictionary](#)<[string](#), [jsonData](#)>,  
[IReadOnlyCollection](#)<[KeyValuePair](#)<[string](#), [jsonData](#)>>,  
[IEnumerable](#)<[KeyValuePair](#)<[string](#), [jsonData](#)>>, [IDictionary](#), [ICollection](#),  
[IEnumerable](#), [IDeserializationCallback](#), [ISerializable](#), [jsonData](#)

## Inherited Members

[Dictionary](#)<[string](#), [jsonData](#)>.Add([string](#), [jsonData](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.Clear() ,  
[Dictionary](#)<[string](#), [jsonData](#)>.ContainsKey([string](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.ContainsValue([jsonData](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.EnsureCapacity([int](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.GetEnumerator() ,  
[Dictionary](#)<[string](#), [jsonData](#)>.OnDeserialization([object](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.Remove([string](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.Remove([string](#), [out jsonData](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.TrimExcess() ,  
[Dictionary](#)<[string](#), [jsonData](#)>.TrimExcess([int](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.TryAdd([string](#), [jsonData](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.TryGetValue([string](#), [out jsonData](#)) ,  
[Dictionary](#)<[string](#), [jsonData](#)>.Comparer , [Dictionary](#)<[string](#), [jsonData](#)>.Count ,  
[Dictionary](#)<[string](#), [jsonData](#)>.this[[string](#)] , [Dictionary](#)<[string](#), [jsonData](#)>.Keys ,  
[Dictionary](#)<[string](#), [jsonData](#)>.Values , [object.Equals\(object\)](#) ,

[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[ICollectionExtensions.ContainsAll<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[ICollectionExtensions.ContainsAll<T>\(ICollection<T>, params T\[\]\)](#) ,  
[ICollectionExtensions.ContainsAny<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[ICollectionExtensions.ContainsAny<T>\(ICollection<T>, params T\[\]\)](#) ,  
[ICollectionExtensions.ContainsNone<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[ICollectionExtensions.ContainsNone<T>\(ICollection<T>, params T\[\]\)](#) ,  
[ICollectionExtensions.ContainsNotAll<T>\(ICollection<T>, IEnumerable<T>\)](#) ,  
[ICollectionExtensions.ContainsNotAll<T>\(ICollection<T>, params T\[\]\)](#) ,  
[IDictionaryExtensions.GetOrAdd<TKey, TValue>\(IDictionary<TKey, TValue>, TKey, TValue\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,

[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Constructors

### Object()

```
public Object()
```

### Object(IDictionary<string, JsonData>)

```
public Object(IDictionary<string, JsonData> collection)
```

#### Parameters

collection [IDictionary<string, JsonData>](#)

### Object(IEnumerable<KeyValuePair<string, JsonData>>)

```
public Object(IEnumerable<KeyValuePair<string, JsonData>> collection)
```

#### Parameters

collection [IEnumerable<KeyValuePair<string, JsonData>>](#)

## Object(int)

```
public Object(int capacity)
```

### Parameters

capacity [int](#)

## Methods

### Add(string, Bool)

```
public void Add(string key, jsonData.Bool value)
```

### Parameters

key [string](#)

value [jsonData.Bool](#)

### Add(string, Float)

```
public void Add(string key, jsonData.Float value)
```

### Parameters

key [string](#)

value [jsonData.Float](#)

### Add(string, Int)

```
public void Add(string key, jsonData.Int value)
```

Parameters

key [string](#)

value [JsonData.Int](#)

## Add(string, List)

```
public void Add(string key, JsonData.List value)
```

Parameters

key [string](#)

value [JsonData.List](#)

## Add(string, Null)

```
public void Add(string key, JsonData.Null value)
```

Parameters

key [string](#)

value [JsonData.Null](#)

## Add(string, Object)

```
public void Add(string key, JsonData.Object value)
```

Parameters

key [string](#)

value [JsonData.Object](#)

## Add(string, String)

```
public void Add(string key, jsonData.String value)
```

### Parameters

key [string](#)

value [jsonData.String](#)

## Add(string, bool)

```
public void Add(string key, bool value)
```

### Parameters

key [string](#)

value [bool](#)

## Add(string, int)

```
public void Add(string key, int value)
```

### Parameters

key [string](#)

value [int](#)

## Add(string, float)

```
public void Add(string key, float value)
```

### Parameters

key [string](#)

value [float](#)

## Add(string, string)

```
public void Add(string key, string value)
```

### Parameters

key [string](#)

value [string](#)

## Append(params Object[])

Returns a JSON object with this object's entries, followed by the first objects's entries, then the second objects's entries, etc.

```
public JsonData.Object Append(params JsonData.Object[] jsonObjs)
```

### Parameters

jsonObjs [Object](#)[]

### Returns

[JsonData.Object](#)

## Append(IEnumerable<Object>)

Returns a JSON object with this object's entries, followed by the first objects's entries, then the second objects's entries, etc.

```
public JsonData.Object Append(IEnumerable<JsonData.Object> jsonObjs)
```

Parameters

jsonObjs [IEnumerable<JsonData.Object>](#)

Returns

[JsonData.Object](#)

## Concat(params Object[])

Combines the JSON objects into one by putting the first object's entries first, then the second object's entries, etc. Returns an empty object if no JSON objects are given.

```
public static JsonData.Object Concat(params JsonData.Object[] jsonObjs)
```

Parameters

jsonObjs [Object\[\]](#)

Returns

[JsonData.Object](#)

## Concat(IEnumerable<Object>)

Combines the JSON objects into one by putting the first object's entries first, then the second object's entries, etc. Returns an empty object if no JSON objects are given.

```
public static JsonData.Object Concat(IEnumerable<JsonData.Object> jsonObjs)
```

Parameters

jsonObjs [IEnumerable<JsonData.Object>](#)

Returns

[JsonData.Object](#)

## Parse(TextReader, bool)

Reads the TextReader, starting at the given index, and attempts to parse it as a JsonData.Object. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData.Object Parse(TextReader reader, bool mustReadAll)
```

### Parameters

reader [TextReader](#)

mustReadAll [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

### Returns

[JsonData.Object](#)

## Parse(string)

Attempts to parse the string into a JsonData.Object.

```
public static JsonData.Object Parse(string str)
```

### Parameters

str [string](#)

### Returns

[JsonData.Object](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as a JsonData.Object. If successful, the index will be moved to the closing curly bracket.

```
public static JsonData.Object Parse(string str, ref int index)
```

Parameters

str [string](#)

index [int](#)

Returns

[JsonData.Object](#)

## ParseMaybeNull(string)

Attempts to parse the string into a JSON object, with the potential to parse it into `JsonData.Null`.

```
public static JsonData ParseMaybeNull(string str)
```

Parameters

str [string](#)

Returns

[JsonData](#)

## ParseMaybeNull(string, ref int)

Attempts to parse the string into a JSON object, with the potential to parse it into `JsonData.Null`. If successful, moves the index to the last parsed character.

```
public static JsonData ParseMaybeNull(string str, ref int index)
```

Parameters

str [string](#)

[index](#) [int ↴](#)

Returns

[JsonData](#)

## Prepend(params Object[])

Returns a JSON object with the first objects's entries, then the second objects's entries, etc., then this object's entries.

```
public JsonData.Object Prepend(params JsonData.Object[] jsonObjs)
```

Parameters

jsonObjs [Object](#)[]

Returns

[JsonData.Object](#)

## Prepend(IEnumerable<Object>)

Returns a JSON object with the first objects's entries, then the second objects's entries, etc., then this object's entries.

```
public JsonData.Object Prepend(IEnumerable<JsonData.Object> jsonObjs)
```

Parameters

jsonObjs [IEnumerable](#)<[JsonData.Object](#)>

Returns

[JsonData.Object](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
public string ToJsonObject(bool pretty)
```

Parameters

pretty bool

If true, puts the string into pretty form - e.g. with indentations.

Returns

string

# Class jsonData.String

Namespace: [PAC.Json](#)

Represents a non-null string in JSON data.

```
public class jsonData.String : jsonData
```

## Inheritance

[object](#) ← jsonData.String

## Implements

[jsonData](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### String(string)

```
public String(string value)
```

## Parameters

value [string](#)

## Properties

### value

```
public string value { get; set; }
```

## Property Value

[string](#)

## Methods

### MaybeNull(string)

Creates a new `JsonData.String` with the given value, unless value is null in which case it returns `JsonData.Null`. Useful since `JsonData.String` cannot hold null strings.

```
public static JsonData MaybeNull(string value)
```

#### Parameters

`value` [string](#)

#### Returns

[JsonData](#)

### Parse(TextReader, bool)

Reads the `TextReader`, starting at the given index, and attempts to parse it as a `JsonData.String`. If successful, the last read character will be the last character of the parsed JSON.

```
public static JsonData.String Parse(TextReader reader, bool mustReadAll)
```

#### Parameters

`reader` [TextReader](#)

`mustReadAll` [bool](#)

If true, it will check that all the data in the reader was used when parsing. Setting this to false is useful for parsing JSON data inside a list or object.

Returns

[JsonData.String](#)

## Parse(string)

Attempts to parse the string into a JsonData.String.

```
public static JsonData.String Parse(string str)
```

Parameters

str [string](#)

Returns

[JsonData.String](#)

## Parse(string, ref int)

Reads the string, starting at the given index, and attempts to parse the string as a JsonData.String. If successful, the index will be moved to the closing quotation mark.

```
public static JsonData.String Parse(string str, ref int index)
```

Parameters

str [string](#)

index [int](#)

Returns

[JsonData.String](#)

## ParseMaybeNull(string)

Attempts to parse the string into a JSON string, with the potential to parse it into `JsonData.Null`.

```
public static JsonData ParseMaybeNull(string str)
```

Parameters

`str` [string](#)

Returns

[JsonData](#)

## ParseMaybeNull(string, ref int)

Attempts to parse the string into a JSON string, with the potential to parse it into `JsonData.Null`. If successful, moves the index to the last parsed character.

```
public static JsonData ParseMaybeNull(string str, ref int index)
```

Parameters

`str` [string](#)

`index` [int](#)

Returns

[JsonData](#)

## ToJsonObject(bool)

Converts it into a string in JSON format.

```
public string ToJsonObject(bool pretty)
```

## Parameters

`pretty` [bool](#)

If true, puts the string into pretty form - e.g. with indentations.

## Returns

[string](#)

# Operators

## implicit operator string(String)

```
public static implicit operator string(JsonData.String jsonString)
```

## Parameters

`jsonString` [JsonData.String](#)

## Returns

[string](#)

## implicit operator String(string)

```
public static implicit operator JsonData.String(string value)
```

## Parameters

`value` [string](#)

## Returns

[JsonData.String](#)

# Namespace PAC.KeyboardShortcuts

## Classes

### [CustomKeyCode](#)

A class to supersede Unity's KeyCode enum to allow keycodes to refer to multiple keys - e.g. Shift instead of separated into LeftShift and RightShift.

### [CustomKeyCode.JsonConverter](#)

Custom JSON converter for CustomKeyCode.

### [KeyboardShortcut](#)

A class to represent a single keyboard shortcut.

### [KeyboardShortcut.JsonConverter](#)

Custom JSON converter for KeyboardShortcut.

### [KeyboardShortcutOptionsManager](#)

Handles the window where you can view / change keyboard shortcuts.

### [KeyboardShortcuts](#)

A class for storing, loading and accessing keyboard shortcuts for defined actions.

# Class CustomKeyCode

Namespace: [PAC.KeyboardShortcuts](#)

A class to supersede Unity's KeyCode enum to allow keycodes to refer to multiple keys - e.g. Shift instead of separated into LeftShift and RightShift.

```
public class CustomKeyCode : IEnumerable
```

## Inheritance

[object](#) ← CustomKeyCode

## Implements

[IEnumerable](#)

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### Alt

```
public static readonly CustomKeyCode Alt
```

### Field Value

[CustomKeyCode](#)

### Ctrl

```
public static readonly CustomKeyCode Ctrl
```

Field Value

[CustomKeyCode](#)

## GreaterThan

```
public static readonly CustomKeyCode GreaterThan
```

Field Value

[CustomKeyCode](#)

## LessThan

```
public static readonly CustomKeyCode LessThan
```

Field Value

[CustomKeyCode](#)

## Minus

```
public static readonly CustomKeyCode Minus
```

Field Value

[CustomKeyCode](#)

## Plus

```
public static readonly CustomKeyCode Plus
```

Field Value

[CustomKeyCode](#)

## Shift

```
public static readonly CustomKeyCode Shift
```

Field Value

[CustomKeyCode](#)

\_0

The keycode for 0.

```
public static readonly CustomKeyCode _0
```

Field Value

[CustomKeyCode](#)

\_1

The keycode for 1.

```
public static readonly CustomKeyCode _1
```

Field Value

[CustomKeyCode](#)

\_2

The keycode for 2.

```
public static readonly CustomKeyCode _2
```

Field Value

[CustomKeyCode](#)

\_3

The keycode for 3.

```
public static readonly CustomKeyCode _3
```

Field Value

[CustomKeyCode](#)

\_4

The keycode for 4.

```
public static readonly CustomKeyCode _4
```

Field Value

[CustomKeyCode](#)

\_5

The keycode for 5.

```
public static readonly CustomKeyCode _5
```

Field Value

[CustomKeyCode](#)

\_6

The keycode for 6.

```
public static readonly CustomKeyCode _6
```

Field Value

[CustomKeyCode](#)

\_7

The keycode for 7.

```
public static readonly CustomKeyCode _7
```

Field Value

[CustomKeyCode](#)

\_8

The keycode for 8.

```
public static readonly CustomKeyCode _8
```

Field Value

[CustomKeyCode](#)

\_9

The keycode for 9.

```
public static readonly CustomKeyCode _9
```

Field Value

[CustomKeyCode](#)

## allKeyCodes

All Unity KeyCodes and all combined keycodes.

```
public static readonly CustomKeyCode[] allKeyCodes
```

Field Value

[CustomKeyCode\[\]](#)

## alphabet

The keycodes A-Z.

```
public static readonly CustomKeyCode[] alphabet
```

Field Value

[CustomKeyCode\[\]](#)

## combinedKeyCodes

All the defined keycodes that combine multiple Unity KeyCodes.

```
public static readonly CustomKeyCode[] combinedKeyCodes
```

Field Value

[CustomKeyCode\[\]](#)

## digits

The keycodes 0-9.

```
public static readonly CustomKeyCode[] digits
```

Field Value

[CustomKeyCode\[\]](#)

## Properties

### displayName

What will be returned from `ToString()`.

```
public string displayName { get; }
```

Property Value

[string](#)

### keyCodes

The Unity KeyCodes comprising this keycode.

```
public List<KeyCode> keyCodes { get; }
```

Property Value

[List](#)<KeyCode>

## Methods

### Contains(KeyCode)

Returns true if the Unity KeyCode forms part of this CustomKeyCode.

```
public bool Contains(KeyCode keyCode)
```

Parameters

keyCode KeyCode

Returns

[bool](#)

## Equals(object)

Determines whether the specified object is equal to the current object.

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

The object to compare with the current object.

Returns

[bool](#)

[true](#) if the specified object is equal to the current object; otherwise, [false](#).

## FromString(string)

Converts a string into the keycode with that display name. Returns null if there isn't one.

```
public static CustomKeyCode FromString(string displayName)
```

Parameters

displayName [string](#)

Returns

[CustomKeyCode](#)

## GetEnumerator()

Returns an enumerator that iterates through a collection.

```
public IEnumatorator GetEnumerator()
```

Returns

[IEnumerator](#)

An [IEnumerator](#) object that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## GetKey()

Returns true while this key is being held down.

```
public bool GetKey()
```

Returns

[bool](#)

## GetKeyDown()

Returns true the frame this key is pressed.

```
public bool GetKeyDown()
```

Returns

[bool](#)

## GetKeyUp()

Returns true the frame this key is released.

```
public bool GetKeyUp()
```

Returns

[bool](#)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Operators

## operator ==(CustomKeyCode, CustomKeyCode)

```
public static bool operator ==(CustomKeyCode keyCode1, CustomKeyCode keyCode2)
```

Parameters

keyCode1 [CustomKeyCode](#)

keyCode2 [CustomKeyCode](#)

Returns

[bool](#)

## implicit operator CustomKeyCode(KeyCode)

```
public static implicit operator CustomKeyCode(KeyCode keyCode)
```

Parameters

keyCode KeyCode

Returns

[CustomKeyCode](#)

## operator !=(CustomKeyCode, CustomKeyCode)

```
public static bool operator !=(CustomKeyCode keyCode1, CustomKeyCode keyCode2)
```

Parameters

keyCode1 [CustomKeyCode](#)

keyCode2 [CustomKeyCode](#)

Returns

[bool](#) ↗

# Class CustomKeyCode.JsonConverter

Namespace: [PAC.KeyboardShortcuts](#)

Custom JSON converter for CustomKeyCode.

```
public class CustomKeyCode.JsonConverter :  
JsonConversion.JsonConverter<CustomKeyCode, JsonData.String>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<CustomKeyCode, JsonData.String>](#) ←  
CustomKeyCode.JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<CustomKeyCode, JsonData.String>.ToJson\(CustomKeyCode\)](#),  
,

[JsonConversion.JsonConverter<CustomKeyCode, JsonData.String>.FromJson\(JsonData.String\)](#),  
[JsonConversion.JsonConverter<CustomKeyCode, JsonData.String>.FromJson\(JsonData\)](#),  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Methods

### FromJson(String)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override CustomKeyCode FromJson(JsonData.String jsonData)
```

Parameters

`jsonData JsonData.String`

Returns

[CustomKeyCode](#)

## ToJson(CustomKeyCode)

Attempts to convert the C# object into JSON data.

```
public override JsonData.String ToJson(CustomKeyCode keyCode)
```

Parameters

`keyCode CustomKeyCode`

Returns

[JsonData.String](#)

# Class KeyboardShortcut

Namespace: [PAC.KeyboardShortcuts](#)

A class to represent a single keyboard shortcut.

```
public class KeyboardShortcut
```

## Inheritance

[object](#) ← KeyboardShortcut

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### KeyboardShortcut(params CustomKeyCode[])

Creates a shortcut from the given keycodes. Sorts the keycodes into the order they would be read.

```
public KeyboardShortcut(params CustomKeyCode[] keyCodes)
```

## Parameters

keyCodes [CustomKeyCode\[\]](#)

## Properties

None

The empty keyboard shortcut - i.e. no keycodes.

```
public static KeyboardShortcut None { get; }
```

Property Value

[KeyboardShortcut](#)

## keyCodes

The keycodes in this shortcut, kept in the order they would be read.

```
public List<CustomKeyCode> keyCodes { get; }
```

Property Value

[List](#)<[CustomKeyCode](#)>

## Methods

### Add(CustomKeyCode)

Adds the keycode to the shortcut. Returns false if it was already in the shortcut; otherwise returns true.

```
public bool Add(CustomKeyCode keyCode)
```

Parameters

keyCode [CustomKeyCode](#)

Returns

[bool](#)

### Contains(CustomKeyCode)

Returns true if the keycode is in the shortcut.

```
public bool Contains(CustomKeyCode keyCode)
```

Parameters

keyCode [CustomKeyCode](#)

Returns

[bool](#)

## Equals(object)

Determines whether the specified object is equal to the current object.

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

The object to compare with the current object.

Returns

[bool](#)

[true](#) if the specified object is equal to the current object; otherwise, [false](#).

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## GetKeys()

Returns true while all keys are being held down.

```
public bool GetKeys()
```

Returns

[bool](#)

## GetKeysDown()

Returns true the frame the final key is pressed.

```
public bool GetKeysDown()
```

Returns

[bool](#)

## GetKeysUp()

Returns true the frame (any key of) this shortcut is released.

```
public bool GetKeysUp()
```

Returns

[bool](#)

## Remove(CustomKeyCode)

Removes the keycode from the shortcut. Returns false if the keycode already wasn't in the shortcut; otherwise returns true.

```
public bool Remove(CustomKeyCode keyCode)
```

Parameters

keyCode [CustomKeyCode](#)

Returns

[bool](#)

## ToArray()

Returns an array of the keycodes in this shortcut, in the order they would be read.

```
public CustomKeyCode[] ToArray()
```

Returns

[CustomKeyCode\[\]](#)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Operators

## operator ==(KeyboardShortcut, KeyboardShortcut)

Checks if the shortcuts have the same set of keycodes.

```
public static bool operator ==(KeyboardShortcut shortcut1,  
KeyboardShortcut shortcut2)
```

Parameters

`shortcut1` [KeyboardShortcut](#)

`shortcut2` [KeyboardShortcut](#)

Returns

[bool](#)

## operator !=(KeyboardShortcut, KeyboardShortcut)

Checks if the shortcuts do not have the same set of keycodes.

```
public static bool operator !=(KeyboardShortcut shortcut1,  
KeyboardShortcut shortcut2)
```

Parameters

`shortcut1` [KeyboardShortcut](#)

`shortcut2` [KeyboardShortcut](#)

Returns

[bool](#)

# Class KeyboardShortcut.JsonConverter

Namespace: [PAC.KeyboardShortcuts](#)

Custom JSON converter for KeyboardShortcut.

```
public class KeyboardShortcut.JsonConverter :  
JsonConversion.JsonConverter<KeyboardShortcut, JsonData.List>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<KeyboardShortcut, JsonData.List>](#) ←  
KeyboardShortcut.JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<KeyboardShortcut, JsonData.List>.ToJson\(KeyboardShortcut\)](#),  
,

[JsonConversion.JsonConverter<KeyboardShortcut, JsonData.List>.FromJson\(JsonData.List\)](#) ,  
[JsonConversion.JsonConverter<KeyboardShortcut, JsonData.List>.FromJson\(JsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Methods

### FromJson(List)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override KeyboardShortcut FromJson(JsonData.List jsonData)
```

Parameters

`jsonData` [JsonData.List](#)

Returns

[KeyboardShortcut](#)

## ToJson(KeyboardShortcut)

Attempts to convert the C# object into JSON data.

```
public override JsonData.List ToJson(KeyboardShortcut keyboardShortcut)
```

Parameters

`keyboardShortcut` [KeyboardShortcut](#)

Returns

[JsonData.List](#)

# Class KeyboardShortcutOptionsManager

Namespace: [PAC.KeyboardShortcuts](#)

Handles the window where you can view / change keyboard shortcuts.

```
public class KeyboardShortcutOptionsManager : MonoBehaviour
```

## Inheritance

[object](#) ← KeyboardShortcutOptionsManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,

[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

# Class KeyboardShortcuts

Namespace: [PAC.KeyboardShortcuts](#)

A class for storing, loading and accessing keyboard shortcuts for defined actions.

```
public class KeyboardShortcuts : MonoBehaviour
```

## Inheritance

[object](#) ← KeyboardShortcuts

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Properties

### shortcuts

A key is an action name. A value is a list of the keyboard shortcuts for that action.

```
public static Dictionary<string, List<KeyboardShortcut>> shortcuts { get; }
```

## Property Value

[Dictionary<string, List<KeyboardShortcut>>](#)

## Methods

### AddShortcut(string, params CustomKeyCode[])

Combines the given keycodes into a single shortcut and adds it for the given action.

```
public static void AddShortcut(string actionPerformed, params CustomKeyCode[] keyCodes)
```

## Parameters

`actionName` [string](#)

`keyCodes` [CustomKeyCode](#)[]

## AddShortcut(string, KeyboardShortcut)

Adds the given shortcut for the given action.

```
public static void AddShortcut(string actionName, KeyboardShortcut shortcut)
```

### Parameters

`actionName` [string](#)

`shortcut` [KeyboardShortcut](#)

## ClearShortcutsFor(string)

Removes all shortcuts for the action.

```
public static void ClearShortcutsFor(string actionName)
```

### Parameters

`actionName` [string](#)

## ContainsKey(string)

Returns true if the action is defined for having keyboard shortcuts.

```
public static bool ContainsKey(string actionName)
```

### Parameters

`actionName` [string](#)

Returns

[bool](#)

## ContainsShortcutFor(string)

Returns true if the action has a shortcut.

```
public static bool ContainsShortcutFor(string actionPerformed)
```

Parameters

actionName [string](#)

Returns

[bool](#)

## GetShortcutsFor(string)

Returns all shortcuts for the action.

```
public static List<KeyboardShortcut> GetShortcutsFor(string actionPerformed)
```

Parameters

actionName [string](#)

Returns

[List](#)<[KeyboardShortcut](#)>

## LoadShortcuts()

Loads saved shortcuts from the disk.

```
public static void LoadShortcuts()
```

## SaveShortcuts()

Saves the current assignment of each shortcut to the disk.

```
public static void SaveShortcuts()
```

## SetShortcut(string, KeyboardShortcut)

Removes any existing shortcuts for the action then adds the given shortcut.

```
public static void SetShortcut(string actionName, KeyboardShortcut shortcut)
```

### Parameters

actionName [string](#)

shortcut [KeyboardShortcut](#)

# Namespace PAC.Layers

## Classes

### [Layer](#)

An abstract class to define what each type of layer must have.

### [Layer.JsonConverter](#)

### [LayerManager](#)

### [LayerTile](#)

### [NormalLayer](#)

A class to represent a normal layer - one that can be drawn on as a regular image.

### [NormalLayer.JsonConverter](#)

### [TileLayer](#)

A class to represent a tile layer - one for placing and editing tilesheet tiles.

### [TileLayer.JsonConverter](#)

## Enums

### [AnimFrameRefMode](#)

### [LayerType](#)

# Enum AnimFrameRefMode

Namespace: [PAC.Layers](#)

```
public enum AnimFrameRefMode
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

MostRecentKeyFrame = 1

NewKeyFrame = 0

# Class Layer

Namespace: [PAC.Layers](#)

An abstract class to define what each type of layer must have.

```
public abstract class Layer
```

## Inheritance

[object](#) ← Layer

## Derived

[NormalLayer](#), [TileLayer](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### Layer(Layer)

```
public Layer(Layer layer)
```

#### Parameters

layer [Layer](#)

### Layer(string, Texture2D)

```
public Layer(string name, Texture2D texture)
```

## Parameters

`name` [string](#)

`texture` [Texture2D](#)

## Fields

### \_blendMode

`protected BlendMode _blendMode`

#### Field Value

[BlendMode](#)

### \_visible

`protected bool _visible`

#### Field Value

[bool](#)

### onPixelsChanged

Called when the SetPixel() or SetPixels() has been called. Passes the array of pixels changed and the frames it was called on.

`protected UnityEvent<IEnumerable<IntVector2>, int[]> onPixelsChanged`

#### Field Value

[UnityEvent<IEnumerable<IntVector2>, int\[\]>](#)

# Properties

## this[int]

Returns the texture of the most recent key frame before or at the frame index. Same as GetTexture(frame).

```
public AnimationKeyFrame this[int frame] { get; }
```

### Parameters

frame [int](#)

### Property Value

[AnimationKeyFrame](#)

## blendMode

```
public BlendMode blendMode { get; set; }
```

### Property Value

[BlendMode](#)

## height

```
public int height { get; protected set; }
```

### Property Value

[int](#)

## keyFrameIndices

The frame indices of the key frames in the animation, in order.

```
public int[] keyFrameIndices { get; }
```

Property Value

[int](#)[]

## keyFrameTextures

The textures of the key frames in the animation, in order.

```
public Texture2D[] keyFrameTextures { get; }
```

Property Value

Texture2D[]

## keyFrames

The key frames of the animation, in order.

```
public List<AnimationKeyFrame> keyFrames { get; protected set; }
```

Property Value

[List](#)<[AnimationKeyFrame](#)>

## layerType

```
public abstract LayerType layerType { get; }
```

Property Value

[LayerType](#)

## locked

```
public bool locked { get; set; }
```

### Property Value

[bool](#)

## name

```
public string name { get; set; }
```

### Property Value

[string](#)

## opacity

```
public float opacity { get; set; }
```

### Property Value

[float](#)

## rect

```
public IntRect rect { get; }
```

### Property Value

[IntRect](#)

## visible

```
public bool visible { get; set; }
```

Property Value

[bool](#)

width

```
public int width { get; protected set; }
```

Property Value

[int](#)

## Methods

### AddKeyFrame(AnimationKeyFrame)

Adds the given key frame. Returns true if it replaces an existing key frame, and false otherwise.

```
protected bool AddKeyFrame(AnimationKeyFrame keyFrame)
```

Parameters

keyFrame [AnimationKeyFrame](#)

Returns

[bool](#)

### AddKeyFrame(int)

Adds a key frame at frame frame. The texture will be that of the most recent key frame. Returns true if it replaces an existing key frame, and false otherwise.

```
protected bool AddKeyFrame(int frame)
```

Parameters

frame [int](#)

Returns

[bool](#)

## AddKeyFrame(int, Texture2D)

Adds a key frame with the given texture at frame frame. Returns true if it replaces an existing key frame, and false otherwise.

```
protected bool AddKeyFrame(int frame, Texture2D texture)
```

Parameters

frame [int](#)

texture [Texture2D](#)

Returns

[bool](#)

## ClearFrames()

Deletes all key frames.

```
public abstract void ClearFrames()
```

## DeepCopy()

Creates a deep copy of the Layer.

```
public abstract Layer DeepCopy()
```

Returns

[Layer](#)

## DeleteKeyFrame(AnimationKeyFrame)

Deletes the given key frame, if it's in the animation, in which case it returns true. Otherwise it returns false.

```
public bool DeleteKeyFrame(AnimationKeyFrame keyFrame)
```

Parameters

keyFrame [AnimationKeyFrame](#)

Returns

[bool](#)

## DeleteKeyFrame(int)

```
public AnimationKeyFrame DeleteKeyFrame(int keyFrame)
```

Parameters

keyFrame [int](#)

Returns

[AnimationKeyFrame](#)

## DeleteKeyFrameNoEvent(int)

Deletes the key frame at the given frame index, if there is one, in which case it returns that key frame. Otherwise it returns null.

```
protected abstract AnimationKeyFrame DeleteKeyFrameNoEvent(int keyframe)
```

Parameters

keyframe [int](#)

Returns

[AnimationKeyFrame](#)

## DeleteMostRecentKeyFrame(int)

Deletes the most recent key frame before or at the given frame index and returns that key frame.

```
public AnimationKeyFrame DeleteMostRecentKeyFrame(int frame)
```

Parameters

frame [int](#)

Returns

[AnimationKeyFrame](#)

## Extend(int, int, int, int)

Extends the dimensions of the layer in each direction by the given amounts.

```
public void Extend(int left, int right, int up, int down)
```

Parameters

left [int](#)

right [int](#)

up [int](#)

down [int](#)

## ExtendNoEvent(int, int, int, int)

Extends the dimensions of the layer in each direction by the given amounts, but does not invoke the onPixelsChanged event.

```
protected abstract void ExtendNoEvent(int left, int right, int up, int down)
```

### Parameters

left [int](#)

right [int](#)

up [int](#)

down [int](#)

## Flip(FlipAxis)

Flips the layer.

```
public void Flip(FlipAxis axis)
```

### Parameters

axis [FlipAxis](#)

## FlipNoEvent(FlipAxis)

Flips the layer, but does not invoke the onPixelsChanged event.

```
protected abstract void FlipNoEvent(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

## GetJsonConverterSet(SemanticVersion)

```
public static JsonConversion.JsonConverterSet GetJsonConverterSet(SemanticVersion  
fromJsonFileFormatVersion)
```

Parameters

fromJsonFileFormatVersion [SemanticVersion](#)

Returns

[JsonConversion.JsonConverterSet](#)

## GetKeyFrame(int)

Returns the most recent key frame before or at the frame index - i.e. the key frame the animation will play at the frame index.

```
public AnimationKeyFrame GetKeyFrame(int frame)
```

Parameters

frame [int](#)

Returns

[AnimationKeyFrame](#)

## GetPixel(IntVector2, int, bool)

Gets the colour of the pixel.

```
public abstract Color GetPixel(IntVector2 pixel, int frame, bool useLayerOpacity = true)
```

Parameters

pixel [IntVector2](#)

frame [int](#)

useLayerOpacity [bool](#)

Returns

Color

## GetPixel(int, int, int, bool)

Gets the colour of the pixel (x, y).

```
public Color GetPixel(int x, int y, int frame, bool useLayerOpacity = true)
```

Parameters

x [int](#)

y [int](#)

frame [int](#)

useLayerOpacity [bool](#)

Returns

Color

## HasKeyFrameAt(int)

Returns whether or not there is a key frame at the given frame index.

```
public bool HasKeyFrameAt(int frame)
```

Parameters

frame [int](#)

Returns

[bool](#)

## IsBlank()

Returns true if and only if all pixels are completely transparent.

```
public bool IsBlank()
```

Returns

[bool](#)

## Rotate(RotationAngle)

Rotates the layer. Rotation is clockwise.

```
public void Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

## RotateNoEvent(RotationAngle)

Rotates the layer, but does not invoke the onPixelsChanged event. Rotation is clockwise.

```
protected abstract void RotateNoEvent(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

## Scale(int, int)

Resizes the dimensions of the file by the scale factor.

```
public void Scale(int newWidth, int newHeight)
```

Parameters

newWidth [int](#)

newHeight [int](#)

## Scale(float)

Resizes the dimensions of the file by the scale factor.

```
public void Scale(float scaleFactor)
```

Parameters

scaleFactor [float](#)

## Scale(float, float)

Resizes the dimensions of the file by the scale factors.

```
public void Scale(float xScaleFactor, float yScaleFactor)
```

Parameters

xScaleFactor [float](#)

yScaleFactor [float](#)

## ScaleNoEvent(int, int)

Resizes the dimensions to the new width and height, but does not invoke the onPixelsChanged event.

```
protected abstract void ScaleNoEvent(int newWidth, int newHeight)
```

### Parameters

newWidth [int](#)

newHeight [int](#)

## ScaleNoEvent(float, float)

Resizes the dimensions of the file by the scale factors, but does not invoke the onPixelsChanged event.

```
protected abstract void ScaleNoEvent(float xScaleFactor, float yScaleFactor)
```

### Parameters

xScaleFactor [float](#)

yScaleFactor [float](#)

## SetPixel(IntVector2, int, Color, AnimFrameRefMode)

Sets the colour of the pixel. Throws an error if the pixel is outside the layer.

```
public IEnumerable<IntVector2> SetPixel(IntVector2 pixel, int frame, Color colour, AnimFrameRefMode frameRefMode)
```

Parameters

pixel [IntVector2](#)

frame [int](#)

colour Color

frameRefMode [AnimFrameRefMode](#)

Returns

[IEnumerable](#)<[IntVector2](#)>

## SetPixel(int, int, int, Color, AnimFrameRefMode)

Sets the colour of the pixel (x, y). Throws an error if the pixel is outside the layer.

```
public IEnumerable<IntVector2> SetPixel(int x, int y, int frame, Color colour,  
AnimFrameRefMode frameRefMode)
```

Parameters

x [int](#)

y [int](#)

frame [int](#)

colour Color

frameRefMode [AnimFrameRefMode](#)

Returns

[IEnumerable](#)<[IntVector2](#)>

## SetPixels(IEnumerable<IntVector2>, int, Color, AnimFrameRefMode)

Sets the colour of the pixels. Throws an error if a pixel is outside the layer.

```
public IEnumerable<IntVector2> SetPixels(IEnumerable<IntVector2> pixels, int frame,  
Color colour, AnimFrameRefMode frameRefMode)
```

Parameters

`pixels` [IEnumerable<IntVector2>](#)

`frame` [int](#)

`colour` [Color](#)

`frameRefMode` [AnimFrameRefMode](#)

Returns

[IEnumerable<IntVector2>](#)

## SetPixelsNoEvent(IEnumerable<IntVector2>, int, Color, AnimFrameRefMode)

Sets the colour of the pixels. You do not need to check the pixels are in the layer as this check is done in Layer.SetPixels(), which is the only way this method is called.

```
protected abstract IEnumerable<IntVector2> SetPixelsNoEvent(IEnumerable<IntVector2>  
pixels, int frame, Color colour, AnimFrameRefMode frameRefMode)
```

Parameters

`pixels` [IEnumerable<IntVector2>](#)

`frame` [int](#)

`colour` [Color](#)

`frameRefMode` [AnimFrameRefMode](#)

Returns

[IEnumerable](#)<IntVector2>

## SubscribeToOnBlendModeChanged(UnityAction)

```
public void SubscribeToOnBlendModeChanged(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToOnKeyFrameAdded(UnityAction<int>)

```
public void SubscribeToOnKeyFrameAdded(UnityAction<int> call)
```

### Parameters

**call** UnityAction<int>

## SubscribeToOnKeyFrameRemoved(UnityAction<int>)

```
public void SubscribeToOnKeyFrameRemoved(UnityAction<int> call)
```

### Parameters

**call** UnityAction<int>

## SubscribeToOnPixelsChanged(UnityAction<IEnumerable<IntVector2>, int[]>)

```
public void SubscribeToOnPixelsChanged(UnityAction<IEnumerable<IntVector2>, int[]> call)
```

### Parameters

`call` UnityAction<IEnumerable<IntVector2>, int[]>

## SubscribeToOnVisibilityChanged(UnityAction)

`public void SubscribeToOnVisibilityChanged(UnityAction call)`

### Parameters

`call` UnityAction

# Class Layer.JsonConverter

Namespace: [PAC.Layers](#)

```
public class Layer.JsonConverter : JsonConversion.JsonConverter<Layer,  
JsonData.Object>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<Layer, jsonData.Object>](#) ← [Layer.JsonConverter](#)

## Inherited Members

[JsonConversion.JsonConverter<Layer, jsonData.Object>.ToJson\(Layer\)](#) ,  
[JsonConversion.JsonConverter<Layer, jsonData.Object>.FromJson\(jsonData.Object\)](#) ,  
[JsonConversion.JsonConverter<Layer, jsonData.Object>.FromJson\(jsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Constructors

### JsonConverter(SemanticVersion)

```
public JsonConverter(SemanticVersion fromJsonFileVersionVersion)
```

## Parameters

[fromJsonFileVersionVersion](#) [SemanticVersion](#)

## Methods

### FromJson(Object)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override Layer FromJson(JsonData.Object jsonData)
```

Parameters

jsonData [JsonData.Object](#)

Returns

[Layer](#)

## ToJson(Layer)

This currently can't be used since JsonConversion.ToJson() only works on concrete types, but Layer is abstract so you cannot have an object without concrete type Layer.

```
public override JsonData.Object ToJson(Layer obj)
```

Parameters

obj [Layer](#)

Returns

[JsonData.Object](#)

Exceptions

[NotImplementedException](#)

# Class LayerManager

Namespace: [PAC.Layers](#)

```
public class LayerManager : MonoBehaviour
```

## Inheritance

[object](#) ← LayerManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### layerTypeSprite

```
public Sprite layerTypeSprite
```

Field Value

Sprite

### tileLayerTypeSprite

```
public Sprite tileLayerTypeSprite
```

Field Value

Sprite

## Properties

### selectedLayer

The last layer that was selected.

```
public Layer selectedLayer { get; }
```

Property Value

[Layer](#)

## selectedLayerIndex

The index of the last layer that was selected.

```
public int selectedLayerIndex { get; }
```

Property Value

[int](#)

## selectedLayerIndices

The indices of all selected layers, in increasing order.

```
public int[] selectedLayerIndices { get; }
```

Property Value

[int](#)[]

## selectedLayers

The selected layers, in order (highest layer first etc).

```
public Layer[] selectedLayers { get; }
```

Property Value

## Methods

### AddLayer()

```
public void AddLayer()
```

### AddLayer(Texture2D)

```
public void AddLayer(Texture2D texture)
```

#### Parameters

`texture` Texture2D

### AddTileLayer()

```
public void AddTileLayer()
```

### DuplicateSelectedLayers()

```
public void DuplicateSelectedLayers()
```

### FlattenSelectedLayers()

```
public void FlattenSelectedLayers()
```

### HideAllBut(Layer)

```
public void HideAllBut(Layer layer)
```

## Parameters

layer [Layer](#)

## MoveSelectedLayersDown()

```
public void MoveSelectedLayersDown()
```

## MoveSelectedLayersUp()

```
public void MoveSelectedLayersUp()
```

## OnFileSwitched()

```
public void OnFileSwitched()
```

## OnLayersChanged()

```
public void OnLayersChanged()
```

## RemoveSelectedLayers()

```
public void RemoveSelectedLayers()
```

## SetLayerBlendMode(int, BlendMode)

```
public void SetLayerBlendMode(int layerIndex, BlendMode blendMode)
```

Parameters

layerIndex [int](#)

blendMode [BlendMode](#)

## SetLayerName(int, string)

```
public void SetLayerName(int layerIndex, string layerName)
```

Parameters

layerIndex [int](#)

layerName [string](#)

## SetLayerOpacity(int, float)

```
public void SetLayerOpacity(int layerIndex, float opacity)
```

Parameters

layerIndex [int](#)

opacity [float](#)

## SubscribeToLayerChange(UnityAction)

```
public void SubscribeToLayerChange(UnityAction call)
```

Parameters

call UnityAction

## SubscribeToVisibilityChange(UnityAction)

```
public void SubscribeToVisibilityChange(UnityAction call)
```

### Parameters

**call** UnityAction

## WorldYCoordOfLayerTile(int)

```
public float WorldYCoordOfLayerTile(int layerTileIndex)
```

### Parameters

**layerTileIndex** [int](#)

### Returns

[float](#)

# Class LayerTile

Namespace: [PAC.Layers](#)

```
public class LayerTile : MonoBehaviour
```

## Inheritance

[object](#) ← LayerTile

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### layer

```
public Layer layer { get; }
```

Property Value

[Layer](#)

### selected

```
public bool selected { get; }
```

Property Value

[bool](#)

### tileToggle

```
public UIToggleButton tileToggle { get; }
```

Property Value

[UIToggleButton](#)

## Methods

### SetLayer(Layer)

```
public void SetLayer(Layer layer)
```

Parameters

layer [Layer](#)

### SubscribeToLockChange(UnityAction)

```
public void SubscribeToLockChange(UnityAction call)
```

Parameters

call UnityAction

### SubscribeToNameChange(UnityAction)

```
public void SubscribeToNameChange(UnityAction call)
```

Parameters

call UnityAction

### SubscribeToRightClick(UnityAction)

```
public void SubscribeToRightClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToVisibilityChange(UnityAction)

```
public void SubscribeToVisibilityChange(UnityAction call)
```

Parameters

`call` UnityAction

# Enum LayerType

Namespace: [PAC.Layers](#)

```
public enum LayerType
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Normal = 0

Tile = 1

# Class NormalLayer

Namespace: [PAC.Layers](#)

A class to represent a normal layer - one that can be drawn on as a regular image.

```
public class NormalLayer : Layer
```

## Inheritance

[object](#) ← [Layer](#) ← NormalLayer

## Inherited Members

[Layer.name](#) , [Layer.width](#) , [Layer.height](#) , [Layer.rect](#) , [Layer.\\_visible](#) , [Layer.visible](#) ,  
[Layer.locked](#) , [Layer.opacity](#) , [Layer.\\_blendMode](#) , [Layer.blendMode](#) , [Layer.keyFrames](#) ,  
[Layer.keyFrameIndices](#) , [Layer.keyFrameTextures](#) , [Layer.onPixelsChanged](#) ,  
[Layer.SetPixel\(int, int, int, Color, AnimFrameRefMode\)](#) ,  
[Layer.SetPixel\(IntVector2, int, Color, AnimFrameRefMode\)](#) ,  
[Layer.SetPixels\(IEnumerable<IntVector2>, int, Color, AnimFrameRefMode\)](#) ,  
[Layer.GetPixel\(int, int, int, bool\)](#) , [Layer.IsBlank\(\)](#) , [Layer.Flip\(FlipAxis\)](#) ,  
[Layer.Rotate\(RotationAngle\)](#) , [Layer.Extend\(int, int, int, int\)](#) , [Layer.Scale\(float\)](#) ,  
[Layer.Scale\(float, float\)](#) , [Layer.Scale\(int, int\)](#) , [Layer.this\[int\]](#) , [Layer.GetKeyFrame\(int\)](#) ,  
[Layer.HasKeyFrameAt\(int\)](#) , [Layer.AddKeyFrame\(int\)](#) , [Layer.AddKeyFrame\(int, Texture2D\)](#) ,  
[Layer.AddKeyFrame\(AnimationKeyFrame\)](#) , [Layer.DeleteMostRecentKeyFrame\(int\)](#) ,  
[Layer.DeleteKeyFrame\(int\)](#) , [Layer.DeleteKeyFrame\(AnimationKeyFrame\)](#) ,  
[Layer.SubscribeToOnPixelsChanged\(UnityAction<IEnumerable<IntVector2>, int\[\]>\)](#) ,  
[Layer.SubscribeToOnVisibilityChanged\(UnityAction\)](#) ,  
[Layer.SubscribeToOnBlendModeChanged\(UnityAction\)](#) ,  
[Layer.SubscribeToOnKeyFrameAdded\(UnityAction<int>\)](#) ,  
[Layer.SubscribeToOnKeyFrameRemoved\(UnityAction<int>\)](#) ,  
[Layer.GetJsonConverterSet\(SemanticVersion\)](#) , [object.Equals\(object\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

## NormalLayer(NormalLayer)

Creates a deep copy of the NormalLayer.

```
public NormalLayer(NormalLayer layer)
```

Parameters

layer [NormalLayer](#)

## NormalLayer(int, int)

```
public NormalLayer(int width, int height)
```

Parameters

width [int](#)

height [int](#)

## NormalLayer(string, int, int)

```
public NormalLayer(string name, int width, int height)
```

Parameters

name [string](#)

width [int](#)

height [int](#)

## NormalLayer(string, Texture2D)

```
public NormalLayer(string name, Texture2D texture)
```

## Parameters

`name` [string](#)

`texture` [Texture2D](#)

## NormalLayer(Texture2D)

```
public NormalLayer(Texture2D texture)
```

## Parameters

`texture` [Texture2D](#)

## Properties

### layerType

```
public override LayerType layerType { get; }
```

## Property Value

[LayerType](#)

## Methods

### ClearFrames()

Deletes all key frames.

```
public override void ClearFrames()
```

### DeepCopy()

Creates a deep copy of the Layer.

```
public override Layer DeepCopy()
```

Returns

[Layer](#)

## DeleteKeyFrameNoEvent(int)

Deletes the key frame at the given frame index, if there is one, in which case it returns that key frame. Otherwise it returns null.

```
protected override AnimationKeyFrame DeleteKeyFrameNoEvent(int keyframe)
```

Parameters

keyframe [int](#)

Returns

[AnimationKeyFrame](#)

## ExtendNoEvent(int, int, int, int)

Extends the dimensions of the layer in each direction by the given amounts, but does not invoke the onPixelsChanged event.

```
protected override void ExtendNoEvent(int left, int right, int up, int down)
```

Parameters

left [int](#)

right [int](#)

up [int](#)

down [int](#)

## Flip(int, FlipAxis, AnimFrameRefMode)

Flips the given frame of the layer.

```
public void Flip(int frame, FlipAxis axis, AnimFrameRefMode frameRefMode)
```

Parameters

frame [int](#)

axis [FlipAxis](#)

frameRefMode [AnimFrameRefMode](#)

## FlipNoEvent(FlipAxis)

Flips the layer, but does not invoke the onPixelsChanged event.

```
protected override void FlipNoEvent(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

## GetPixel(IntVector2, int, bool)

Gets the colour of the pixel.

```
public override Color GetPixel(IntVector2 pixel, int frame, bool useLayerOpacity  
= true)
```

Parameters

pixel [IntVector2](#)

frame [int](#)

useLayerOpacity [bool](#)

Returns

Color

## OverlayTexture(int, Texture2D, IntVector2, AnimFrameRefMode)

Overlays the texture onto the given frame, placing the bottom-left corner at the coordinates 'offset' (which don't have to be within the image). Uses Normal blend mode.

```
public void OverlayTexture(int frame, Texture2D overlayTex, IntVector2 offset,  
AnimFrameRefMode frameRefMode)
```

Parameters

frame [int](#)

overlayTex [Texture2D](#)

offset [IntVector2](#)

frameRefMode [AnimFrameRefMode](#)

## OverlayTexture(int, Texture2D, AnimFrameRefMode)

Overlays the texture onto the given frame. Uses Normal blend mode.

```
public void OverlayTexture(int frame, Texture2D overlayTex,  
AnimFrameRefMode frameRefMode)
```

Parameters

frame [int](#)

overlayTex [Texture2D](#)

frameRefMode [AnimFrameRefMode](#)

## OverlayTexture(Texture2D)

Overlays the texture onto every frame. Uses Normal blend mode.

```
public void OverlayTexture(Texture2D overlayTex)
```

### Parameters

overlayTex Texture2D

## OverlayTexture(Texture2D, IntVector2)

Overlays the texture onto every frame, placing the bottom-left corner at the coordinates 'offset' (which don't have to be within the image). Uses Normal blend mode.

```
public void OverlayTexture(Texture2D overlayTex, IntVector2 offset)
```

### Parameters

overlayTex Texture2D

offset [IntVector2](#)

## Rotate(int, RotationAngle, AnimFrameRefMode)

Rotates the given frame of the layer. Rotation is clockwise.

```
public void Rotate(int frame, RotationAngle angle, AnimFrameRefMode frameRefMode)
```

### Parameters

frame [int](#)

angle [RotationAngle](#)

frameRefMode [AnimFrameRefMode](#)

## RotateNoEvent(RotationAngle)

Rotates the layer, but does not invoke the onPixelsChanged event. Rotation is clockwise.

```
protected override void RotateNoEvent(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

## ScaleNoEvent(int, int)

Resizes the dimensions to the new width and height, but does not invoke the onPixelsChanged event.

```
protected override void ScaleNoEvent(int newWidth, int newHeight)
```

Parameters

newWidth [int](#)

newHeight [int](#)

## ScaleNoEvent(float, float)

Resizes the dimensions of the file by the scale factors, but does not invoke the onPixelsChanged event.

```
protected override void ScaleNoEvent(float xScaleFactor, float yScaleFactor)
```

Parameters

xScaleFactor [float](#)

yScaleFactor [float](#)

## SetPixelsNoEvent(IEnumerable<IntVector2>, int, Color, AnimFrameRefMode)

Sets the colour of the pixels. You do not need to check the pixels are in the layer as this check is done in Layer.SetPixels(), which is the only way this method is called.

```
protected override IEnumerable<IntVector2> SetPixelsNoEvent(IEnumerable<IntVector2> pixels, int frame, Color colour, AnimFrameRefMode frameRefMode)
```

### Parameters

`pixels` [IEnumerable<IntVector2>](#)

`frame` [int](#)

`colour` [Color](#)

`frameRefMode` [AnimFrameRefMode](#)

### Returns

[IEnumerable<IntVector2>](#)

## SetTexture(int, Texture2D, AnimFrameRefMode)

Sets the texture at the given frame.

```
public void SetTexture(int frame, Texture2D texture, AnimFrameRefMode frameRefMode)
```

### Parameters

`frame` [int](#)

`texture` [Texture2D](#)

`frameRefMode` [AnimFrameRefMode](#)

# Class NormalLayer.JsonConverter

Namespace: [PAC.Layers](#)

```
public class NormalLayer.JsonConverter : JsonConversion.JsonConverter<NormalLayer,  
JsonData.Object>
```

## Inheritance

[object](#) ← [JsonConversion.JsonConverter<NormalLayer, jsonData.Object>](#) ←  
NormalLayer.JsonConverter

## Inherited Members

[JsonConversion.JsonConverter<NormalLayer, jsonData.Object>.ToJson\(NormalLayer\)](#) ,  
[JsonConversion.JsonConverter<NormalLayer, jsonData.Object>.FromJson\(JsonData.Object\)](#) ,  
[JsonConversion.JsonConverter<NormalLayer, jsonData.Object>.FromJson\(JsonData\)](#) ,  
[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### JsonConverter(SemanticVersion)

```
public JsonConverter(SemanticVersion fromJsonFileVersionVersion)
```

## Parameters

fromJsonFileVersionVersion [SemanticVersion](#)

## Methods

### FromJson(Object)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload `FromJson(JsonData jsonData)` that ensures it is of type `JsonDataType` before calling the specific overload `FromJSON(JsonDataType jsonData)`.

```
public override NormalLayer FromJson(JsonData.Object jsonData)
```

Parameters

`jsonData` [JsonData.Object](#)

Returns

[NormalLayer](#)

## ToJson(NormalLayer)

Attempts to convert the C# object into JSON data.

```
public override JsonData.Object ToJson(NormalLayer layer)
```

Parameters

`layer` [NormalLayer](#)

Returns

[JsonData.Object](#)

# Class TileLayer

Namespace: [PAC.Layers](#)

A class to represent a tile layer - one for placing and editing tileset tiles.

```
public class TileLayer : Layer
```

## Inheritance

[object](#) ← [Layer](#) ← TileLayer

## Inherited Members

[Layer.name](#) , [Layer.width](#) , [Layer.height](#) , [Layer.rect](#) , [Layer.\\_visible](#) , [Layer.visible](#) ,  
[Layer.locked](#) , [Layer.opacity](#) , [Layer.\\_blendMode](#) , [Layer.blendMode](#) , [Layer.keyFrames](#) ,  
[Layer.keyFrameIndices](#) , [Layer.keyFrameTextures](#) , [Layer.onPixelsChanged](#) ,  
[Layer.SetPixel\(int, int, int, Color, AnimFrameRefMode\)](#) ,  
[Layer.SetPixel\(IntVector2, int, Color, AnimFrameRefMode\)](#) ,  
[Layer.SetPixels\(IEnumerable<IntVector2>, int, Color, AnimFrameRefMode\)](#) ,  
[Layer.GetPixel\(int, int, int, bool\)](#) , [Layer.IsBlank\(\)](#) , [Layer.Flip\(FlipAxis\)](#) ,  
[Layer.Rotate\(RotationAngle\)](#) , [Layer.Extend\(int, int, int, int\)](#) , [Layer.Scale\(float\)](#) ,  
[Layer.Scale\(float, float\)](#) , [Layer.Scale\(int, int\)](#) , [Layer.this\[int\]](#) , [Layer.GetKeyFrame\(int\)](#) ,  
[Layer.HasKeyFrameAt\(int\)](#) , [Layer.AddKeyFrame\(int\)](#) , [Layer.AddKeyFrame\(int, Texture2D\)](#) ,  
[Layer.AddKeyFrame\(AnimationKeyFrame\)](#) , [Layer.DeleteMostRecentKeyFrame\(int\)](#) ,  
[Layer.DeleteKeyFrame\(int\)](#) , [Layer.DeleteKeyFrame\(AnimationKeyFrame\)](#) ,  
[Layer.SubscribeToOnPixelsChanged\(UnityAction<IEnumerable<IntVector2>, int\[\]>\)](#) ,  
[Layer.SubscribeToOnVisibilityChanged\(UnityAction\)](#) ,  
[Layer.SubscribeToOnBlendModeChanged\(UnityAction\)](#) ,  
[Layer.SubscribeToOnKeyFrameAdded\(UnityAction<int>\)](#) ,  
[Layer.SubscribeToOnKeyFrameRemoved\(UnityAction<int>\)](#) ,  
[Layer.GetJsonConverterSet\(SemanticVersion\)](#) , [object.Equals\(object\)](#) ,  
[object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,  
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

## TileLayer(TileLayer)

Creates a deep copy of the TileLayer.

```
public TileLayer(TileLayer layer)
```

Parameters

layer [TileLayer](#)

## TileLayer(string, int, int)

```
public TileLayer(string name, int width, int height)
```

Parameters

name [string](#)

width [int](#)

height [int](#)

## Properties

### layerType

```
public override LayerType layerType { get; }
```

Property Value

[LayerType](#)

### tiles

The tiles on this layer.

```
public List<Tile> tiles { get; }
```

Property Value

[List](#)<[Tile](#)>

## Methods

### AddTile(Tile)

Adds the tile to the layer.

```
public void AddTile(Tile tile)
```

### Parameters

tile [Tile](#)

### ClearFrames()

Deletes all key frames.

```
public override void ClearFrames()
```

### ClearTiles()

Removes all tiles.

```
public void ClearTiles()
```

### ContainsTile(Tile)

Returns true if the tile appears on this layer.

```
public bool ContainsTile(Tile tile)
```

Parameters

[tile](#) [Tile](#)

Returns

[bool](#) ↗

## DeepCopy()

Creates a deep copy of the Layer.

```
public override Layer DeepCopy()
```

Returns

[Layer](#)

## DeleteKeyFrameNoEvent(int)

Deletes the key frame at the given frame index, if there is one, in which case it returns that key frame. Otherwise it returns null.

```
protected override AnimationKeyFrame DeleteKeyFrameNoEvent(int keyframe)
```

Parameters

[keyframe](#) [int](#) ↗

Returns

[AnimationKeyFrame](#)

## ExtendNoEvent(int, int, int, int)

Extends the dimensions of the layer in each direction by the given amounts, but does not invoke the onPixelsChanged event.

```
protected override void ExtendNoEvent(int left, int right, int up, int down)
```

### Parameters

left [int](#)

right [int](#)

up [int](#)

down [int](#)

## FlipNoEvent(FlipAxis)

Flips the layer, but does not invoke the onPixelsChanged event.

```
protected override void FlipNoEvent(FlipAxis axis)
```

### Parameters

axis [FlipAxis](#)

## GetLinkedPixels(IntVector2)

Get the pixels that are linked to the given pixel due to multiple tiles having the same file - i.e. they point to the same pixel within the tiles' file.

```
public IEnumerable<IntVector2> GetLinkedPixels(IntVector2 pixel)
```

### Parameters

pixel [IntVector2](#)

Returns

[IEnumerable](#)<IntVector2>

## GetLinkedPixels(IEnumerable<IntVector2>)

Get the pixels that are linked to the given pixels due to multiple tiles having the same file - i.e. they point to the same pixels within the tiles' file.

```
public IEnumerable<IntVector2> GetLinkedPixels(IEnumerable<IntVector2> pixels)
```

Parameters

`pixels` [IEnumerable](#)<IntVector2>

Returns

[IEnumerable](#)<IntVector2>

## GetPixel(IntVector2, int, bool)

Gets the colour of the pixel.

```
public override Color GetPixel(IntVector2 pixel, int frame, bool useLayerOpacity = true)
```

Parameters

`pixel` [IntVector2](#)

`frame` [int](#)

`useLayerOpacity` [bool](#)

Returns

Color

## PixelToTile(IntVector2)

Gets the tile that the pixel lands in, or null if there isn't one.

```
public Tile PixelToTile(IntVector2 pixel)
```

Parameters

`pixel` [IntVector2](#)

Returns

[Tile](#)

## PixelToTile(int, int)

Gets the tile that the pixel (x, y) lands in, or null if there isn't one.

```
public Tile PixelToTile(int x, int y)
```

Parameters

`x` [int](#)

`y` [int](#)

Returns

[Tile](#)

## RemoveTile(Tile)

Removes the tile from the layer. Throws an error if the tile is not in the layer.

```
public void RemoveTile(Tile tile)
```

Parameters

## RerenderKeyFrame(int)

Rerenders the keyframe.

```
public void RerenderKeyFrame(int frame)
```

Parameters

frame [int](#)

## RerenderKeyFrame(int, IntRect)

Rerenders the section of the keyframe within the given rect.

```
public void RerenderKeyFrame(int frame, IntRect rect)
```

Parameters

frame [int](#)

rect [IntRect](#)

## RerenderKeyFrame(int, IEnumerable<IntVector2>)

Rerenders the given pixels of the keyframe.

```
public void RerenderKeyFrame(int frame, IEnumerable<IntVector2> pixels)
```

Parameters

frame [int](#)

pixels [IEnumerable](#)<[IntVector2](#)>

## RerenderKeyFrames()

Rerenders all keyframes.

```
public void RerenderKeyFrames()
```

## RerenderKeyFrames(IntRect)

Rerenders the section of every keyframe within the given rect.

```
public void RerenderKeyFrames(IntRect rect)
```

Parameters

rect [IntRect](#)

## RerenderKeyFrames(IEnumerable<IntVector2>)

Rerenders the given pixels of every keyframe.

```
public void RerenderKeyFrames(IEnumerable<IntVector2> pixels)
```

Parameters

pixels [IEnumerable](#)<[IntVector2](#)>

## RerenderKeyFrames(int[])

Rerenders the given keyframes.

```
public void RerenderKeyFrames(int[] keyFrames)
```

Parameters

keyFrames [int](#)[]

## RerenderKeyFrames(int[], IntRect)

Rerenders the section of the given keyframes within the given rect.

```
public void RerenderKeyFrames(int[] keyFrames, IntRect rect)
```

Parameters

keyFrames [int\[\]](#)

rect [IntRect](#)

## RerenderKeyFrames(int[], IEnumerable<IntVector2>)

Rerenders the given pixels of the given keyframes.

```
public void RerenderKeyFrames(int[] keyFrames, IEnumerable<IntVector2> pixels)
```

Parameters

keyFrames [int\[\]](#)

pixels [IEnumerable<IntVector2>](#)

## RotateNoEvent(RotationAngle)

Rotates the layer, but does not invoke the onPixelsChanged event. Rotation is clockwise.

```
protected override void RotateNoEvent(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

## ScaleNoEvent(int, int)

Resizes the dimensions to the new width and height, but does not invoke the onPixelsChanged event.

```
protected override void ScaleNoEvent(int newWidth, int newHeight)
```

## Parameters

newWidth [int](#)

newHeight [int](#)

## ScaleNoEvent(float, float)

Resizes the dimensions of the file by the scale factors, but does not invoke the onPixelsChanged event.

```
protected override void ScaleNoEvent(float xScaleFactor, float yScaleFactor)
```

## Parameters

xScaleFactor [float](#)

yScaleFactor [float](#)

## SetPixelsNoEvent(IEnumerable<IntVector2>, int, Color, AnimFrameRefMode)

Sets the colour of the pixels. You do not need to check the pixels are in the layer as this check is done in Layer.SetPixels(), which is the only way this method is called.

```
protected override IEnumerable<IntVector2> SetPixelsNoEvent(IEnumerable<IntVector2> pixels, int frame, Color colour, AnimFrameRefMode frameRefMode)
```

## Parameters

pixels [IEnumerable](#)<[IntVector2](#)>

frame [int](#)

`colour` Color

`frameRefMode` [AnimFrameRefMode](#)

Returns

[IEnumerable](#)<[IntVector2](#)>

# Class TileLayer.JsonConverter

Namespace: [PAC.Layers](#)

```
public class TileLayer.JsonConverter : JsonConversion.JsonConverter<TileLayer,  
JsonData.Object>
```

## Inheritance

```
object ← JsonConversion.JsonConverter<TileLayer, jsonData.Object> ←  
TileLayer.JsonConverter
```

## Inherited Members

```
JsonConversion.JsonConverter<TileLayer, jsonData.Object>.ToJson\(TileLayer\) ,  
JsonConversion.JsonConverter<TileLayer, jsonData.Object>.FromJson\(jsonData.Object\) ,  
JsonConversion.JsonConverter<TileLayer, jsonData.Object>.FromJson\(jsonData\) ,  
object.Equals\(object\) , object.Equals\(object, object\) , object.GetHashCode\(\) ,  
object.GetType\(\) , object.MemberwiseClone\(\) , object.ReferenceEquals\(object, object\) ,  
object.ToString\(\)
```

## Extension Methods

```
ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\) ,  
ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)
```

## Constructors

### JsonConverter(SemanticVersion)

```
public JsonConverter(SemanticVersion fromJsonFileVersionVersion)
```

## Parameters

fromJsonFileVersionVersion [SemanticVersion](#)

## Methods

### FromJson(Object)

Attempts to convert the JSON data into a C# object of the given type.

NOTE: JsonConverter has a default implementation for an overload FromJson(JsonData jsonData) that ensures it is of type JsonDataType before calling the specific overload FromJSON(JsonDataType jsonData).

```
public override TileLayer FromJson(JsonData.Object jsonData)
```

Parameters

jsonData [JsonData.Object](#)

Returns

[TileLayer](#)

## ToJson(TileLayer)

Attempts to convert the C# object into JSON data.

```
public override JsonData.Object ToJson(TileLayer layer)
```

Parameters

layer [TileLayer](#)

Returns

[JsonData.Object](#)

# Namespace PAC.Maths

## Classes

### [Combinatorics](#)

A collection of common combinatorial functions, such as computing factorials or generating all permutations of a set.

### [MathExtensions](#)

Defines some mathematical functions.

# Class Combinatorics

Namespace: [PAC.Maths](#)

A collection of common combinatorial functions, such as computing factorials or generating all permutations of a set.

```
public static class Combinatorics
```

## Inheritance

[object](#) ← Combinatorics

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

# Methods

## BinomialCoefficient(int, int)

The coefficient of  $x^k$  in  $(x + 1)^n$ . An alias for [Choose\(int, int\)](#).

```
public static int BinomialCoefficient(int n, int k)
```

### Parameters

n [int](#)

k [int](#)

### Returns

[int](#)

## See Also

[Choose\(int, int\)](#)

## Choose(int, int)

The choose function. An alias for [BinomialCoefficient\(int, int\)](#).

```
public static int Choose(int n, int k)
```

Parameters

n [int](#)

k [int](#)

Returns

[int](#)

### See Also

[BinomialCoefficient\(int, int\)](#)

## Factorial(int)

Computes  $n * (n - 1) * \dots * 2 * 1$ . Returns 1 if n is 0.

```
public static int Factorial(int n)
```

Parameters

n [int](#)

Returns

[int](#)

## FallingFactorial(int, int)

Computes  $n * (n - 1) * \dots * (n - (k - 1))$ . Returns 1 if k is 0.

```
public static int FallingFactorial(int n, int k)
```

Parameters

n [int](#)

k [int](#)

Returns

[int](#)

## MultiChoose(int, int)

```
public static int MultiChoose(int n, int k)
```

Parameters

n [int](#)

k [int](#)

Returns

[int](#)

## NumberOfCombinations(int)

```
public static int NumberOfCombinations(int sizeOfSet)
```

Parameters

sizeOfSet [int](#)

Returns

[int](#)

## NumberOfCombinations(int, int)

```
public static int NumberOfCombinations(int numElementsToChooseFrom,  
int lengthOfCombination)
```

## Parameters

numElementsToChooseFrom [int](#)

lengthOfCombination [int](#)

## Returns

[int](#)

## NumberOfMultisets(int)

```
public static int NumberOfMultisets(int sizeOfSet)
```

## Parameters

sizeOfSet [int](#)

## Returns

[int](#)

## NumberOfMultisets(int, int)

```
public static int NumberOfMultisets(int numElementsToChooseFrom, int sizeOfMultiset)
```

## Parameters

numElementsToChooseFrom [int](#)

sizeOfMultiset [int](#)

## Returns

[int](#)

## NumberOfPermutations(int)

The number of possible permutations of a set of the given size. (Permutations do not allow repeated elements.)

```
public static int NumberOfPermutations(int sizeOfSet)
```

Parameters

sizeOfSet [int](#)

Returns

[int](#)

## NumberOfPermutations(int, int)

The number of possible permutations of the given length with elements in a set of the given size. (Permutations do not allow repeated elements.)

```
public static int NumberOfPermutations(int numElementsToChooseFrom,  
int lengthOfPermutation)
```

Parameters

numElementsToChooseFrom [int](#)

lengthOfPermutation [int](#)

Returns

[int](#)

## NumberOfTuples(int, int)

The number of possible tuples of the given length with elements in a set of the given size.  
(Tuples allow repeated elements.)

```
public static int NumberOfTuples(int numElementsToChooseFrom, int lengthOfTuple)
```

## Parameters

numElementsToChooseFrom [int](#)

lengthOfTuple [int](#)

## Returns

[int](#)

## Pairs<T>(IEnumerable<T>)

Lazily generates all possible pairs (*x*, *y*) with *x* and *y* in the given [IEnumerable<T>](#). *x* and *y* don't have to be distinct.

```
public static IEnumerable<(T first, T second)> Pairs<T>(this IEnumerable<T> set)
```

## Parameters

set [IEnumerable](#)<T>

## Returns

[IEnumerable](#)<(T [first](#), T [second](#))>

## Type Parameters

T

## Remarks

If the given [IEnumerable<T>](#) has duplicate elements, those elements will be treated as though they were distinct.

For a description of the order in which the pairs are yielded, see [Tuples<T>](#) ([IEnumerable<T>](#), [int](#)).

## Pairs<TFirst, TSecond>(IEnumerable<TFirst>, IEnumerable<TSecond>)

Lazily generates all possible pairs (`x`, `y`) with `x` in `first` and `y` in `second`.

```
public static IEnumerable<(TFirst first, TSecond second)> Pairs<TFirst, TSecond>
(this IEnumerable<TFirst> first, IEnumerable<TSecond> second)
```

### Parameters

`first` [IEnumerable](#)<TFirst>

`second` [IEnumerable](#)<TSecond>

### Returns

[IEnumerable](#)<(TFirst `first`, TSecond `second`)>

### Type Parameters

TFirst

TSecond

### Remarks

If a given [IEnumerable](#)<T> has duplicate elements, those elements will be treated as though they were distinct.

For a description of the order in which the pairs are yielded, see [Tuples<T>](#) ([IEnumerable<T>](#), [int](#)).

## Permutations<T>(IEnumerable<T>)

Lazily generates all possible permutations of the elements in the given [IEnumerable](#)<T>. (Permutations do not allow repeated elements.)

```
public static IEnumerable<IEnumerable<T>> Permutations<T>(this  
IEnumerable<T> elements)
```

## Parameters

elements [IEnumerable](#)<T>

## Returns

[IEnumerable](#)<[IEnumerable](#)<T>>

## Type Parameters

T

## Remarks

If the given [IEnumerable](#)<T> has duplicate elements, they will be treated as though they were distinct.

## Permutations<T>(IEnumerable<T>, int)

Lazily generates all possible permutations of the given length with elements in the given [IEnumerable](#)<T>. (Permutations do not allow repeated elements.)

```
public static IEnumerable<IEnumerable<T>> Permutations<T>(this IEnumerable<T>  
elements, int length)
```

## Parameters

elements [IEnumerable](#)<T>

length [int](#)

## Returns

[IEnumerable](#)<[IEnumerable](#)<T>>

## Type Parameters

## Remarks

If the given [IEnumerable<T>](#) has duplicate elements, they will be treated as though they were distinct.

## RisingFactorial(int, int)

Computes  $n * (n + 1) * \dots * (n + (k - 1))$ . Returns 1 if  $k$  is 0.

```
public static int RisingFactorial(int n, int k)
```

## Parameters

$n$  [int](#)

$k$  [int](#)

## Returns

[int](#)

## Triples<T>(IEnumerable<T>)

Lazily generates all possible triples  $(x, y, z)$  with  $x, y$  and  $z$  in the given [IEnumerable<T>](#).  $x, y$  and  $z$  don't have to all be distinct.

```
public static IEnumerable<(T first, T second, T third)> Triples<T>(this
IEnumerable<T> set)
```

## Parameters

$set$  [IEnumerable](#)<T>

## Returns

[IEnumerable](#)<(T [first](#), T [second](#), T [third](#))>

## Type Parameters

T

## Remarks

If the given [IEnumerable<T>](#) has duplicate elements, those elements will be treated as though they were distinct.

For a description of the order in which the triples are yielded, see [Tuples<T>\(IEnumerable<T>, int\)](#).

## Triples<TFirst, TSecond, TThird>(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)

Lazily generates all possible triples (`x`, `y`, `z`) with `x` in `first`, `y` in `second` and `z` in `third`.

```
public static IEnumerable<(TFirst first, TSecond second, TThird third)>
Triples<TFirst, TSecond, TThird>(this IEnumerable<TFirst> first,
IEnumerable<TSecond> second, IEnumerable<TThird> third)
```

## Parameters

`first` [IEnumerable<TFirst>](#)

`second` [IEnumerable<TSecond>](#)

`third` [IEnumerable<TThird>](#)

## Returns

[IEnumerable<\(TFirst first, TSecond second, TThird third\)>](#)

## Type Parameters

`TFirst`

`TSecond`

`TThird`

## Remarks

If a given [IEnumerable<T>](#) has duplicate elements, those elements will be treated as though they were distinct.

For a description of the order in which the triples are yielded, see [Tuples<T>\(IEnumerable<T>, int\)](#).

## Tuples<T>(IEnumerable<T>, int)

Lazily generates all possible tuples of the given length with elements in the given [IEnumerable<T>](#). (Tuples allow repeated elements.)

```
public static IEnumerable<IEnumerable<T>> Tuples<T>(this IEnumerable<T> elements,  
int length)
```

## Parameters

elements [IEnumerable<T>](#)

length [int](#)

## Returns

[IEnumerable<IEnumerable<T>>](#)

## Type Parameters

T

## Remarks

If the given [IEnumerable<T>](#) has duplicate elements, they will be treated as though they were distinct.

The order of the tuples will be lexicographic based on the order in which the elements appear in the input, with the left-most coordinate being the most significant. This is equivalent to using nested [foreach](#) loops, with a loop being less nested corresponding to a further-left coordinate.

For example, the tuples of length 2 with elements in { 0, 1 } will be in the order { { 0, 0 }, { 0, 1 }, { 1, 0 }, { 1, 1 } }. This is equivalent to

```
foreach (int element1 in set)
{
    foreach (int element2 in set)
    {
        yield return new int[] { element1, element2 };
    }
}
```

# Class MathExtensions

Namespace: [PAC.Maths](#)

Defines some mathematical functions.

```
public static class MathExtensions
```

## Inheritance

[object](#) ← MathExtensions

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Methods

### CeilToMultipleOf(float, int)

Returns the least multiple of `multipleOf` that is  $\geq$  `toRound`.

```
public static int CeilToMultipleOf(this float toRound, int multipleOf)
```

#### Parameters

`toRound` [float](#)

`multipleOf` [int](#)

#### Returns

[int](#)

#### Exceptions

[ArgumentException](#)

`multipleOf` is 0.

## CeilToMultipleOf(float, float)

Returns the least multiple of `multipleOf` that is  $\geq$  `toRound`.

```
public static float CeilToMultipleOf(this float toRound, float multipleOf)
```

Parameters

`toRound` [float](#)

`multipleOf` [float](#)

Returns

[float](#)

Exceptions

[ArgumentException](#)

`multipleOf` is 0.

## ClampNonNegative(int)

Clamps the value so it's non-negative.

```
public static int ClampNonNegative(this int x)
```

Parameters

`x` [int](#)

Returns

[int](#)

- `x` if `x`  $\geq 0$
- 0 if `x`  $< 0$

## ClampNonNegative(long)

Clamps the value so it's non-negative.

```
public static long ClampNonNegative(this long x)
```

Parameters

x [long](#)

Returns

[long](#)

- x if x >= 0
- 0 if x < 0

## ClampNonNegative(float)

Clamps the value so it's non-negative.

```
public static float ClampNonNegative(this float x)
```

Parameters

x [float](#)

Returns

[float](#)

- x if x >= 0
- 0 if x < 0

## ClampNonPositive(int)

Clamps the value so it's non-positive.

```
public static int ClampNonPositive(this int x)
```

## Parameters

x [int](#)

## Returns

[int](#)

- x if x <= 0
- 0 if x > 0

## ClampNonPositive(float)

Clamps the value so it's non-positive.

```
public static float ClampNonPositive(this float x)
```

## Parameters

x [float](#)

## Returns

[float](#)

- x if x <= 0
- 0 if x > 0

## FloorToMultipleOf(float, int)

Returns the greatest multiple of `multipleOf` that is <= `toRound`.

```
public static int FloorToMultipleOf(this float toRound, int multipleOf)
```

## Parameters

`toRound` [float](#)

`multipleOf` [int](#)

Returns

[int](#)

Exceptions

[ArgumentException](#)

`multipleOf` is 0.

## FloorToMultipleOf(float, float)

Returns the greatest multiple of `multipleOf` that is  $\leq$  `toRound`.

```
public static float FloorToMultipleOf(this float toRound, float multipleOf)
```

Parameters

`toRound` [float](#)

`multipleOf` [float](#)

Returns

[float](#)

Exceptions

[ArgumentException](#)

`multipleOf` is 0.

## Gcd(int, int)

Computes the greatest non-negative common divisor of `a` and `b`.

```
public static int Gcd(this int a, int b)
```

Parameters

a [int](#)

b [int](#)

Returns

[int](#)

Exceptions

[ArgumentOutOfRangeException](#)

a or b are [MinValue](#).

## Gcd(long, long)

Computes the greatest non-negative common divisor of a and b.

```
public static long Gcd(this long a, long b)
```

Parameters

a [long](#)

b [long](#)

Returns

[long](#)

Exceptions

[ArgumentOutOfRangeException](#)

a or b are [MinValue](#).

## Mod(int, int)

Returns `a mod b`, giving a result in the range  $[0, \text{abs}(b))$ .

```
public static int Mod(this int a, int b)
```

Parameters

`a` [int](#)

`b` [int](#)

Returns

[int](#)

Exceptions

[DivideByZeroException](#)

`b` is 0.

## Mod(long, long)

Returns `a mod b`, giving a result in the range  $[0, \text{abs}(b))$ .

```
public static long Mod(this long a, long b)
```

Parameters

`a` [long](#)

`b` [long](#)

Returns

[long](#)

Exceptions

## [DivideByZeroException](#)

**b** is 0.

## Mod(float, float)

Returns **a mod b**, giving a result in the range [0, abs(b)).

```
public static float Mod(this float a, float b)
```

Parameters

**a** [float](#)

**b** [float](#)

Returns

[float](#)

Exceptions

## [DivideByZeroException](#)

**b** is 0.

## Pow(int, int)

Computes **n ^ exponent** (**n** raised to the **exponent**).

```
public static int Pow(this int n, int exponent)
```

Parameters

**n** [int](#)

**exponent** [int](#)

Returns

[int](#)

## Remarks

$0^0$  is defined to be 1.

## Exceptions

[ArgumentOutOfRangeException](#)

`exponent` is negative.

## Round(float, int)

Rounds `toRound` to `decimalPlaces` decimal places.

```
public static float Round(this float toRound, int decimalPlaces)
```

## Parameters

`toRound` [float](#)

`decimalPlaces` [int](#)

## Returns

[float](#)

## Exceptions

[ArgumentOutOfRangeException](#)

`decimalPlaces` is negative.

## RoundAwayFrom(float, int)

Rounds `toRound` to an integer, choosing to round up or down so that it becomes further (or unchanged) away from `awayFrom`.

```
public static float RoundAwayFrom(this float toRound, int awayFrom)
```

## Parameters

toRound [float](#)

awayFrom [int](#)

## Returns

[float](#)

- `ceil(toRound)` if `toRound >= awayFrom`
- `floor(toRound)` if `toRound < awayFrom`

## RoundToIntAwayFrom(float, int)

Rounds `toRound` to an integer, choosing to round up or down so that it becomes further (or unchanged) away from `awayFrom`.

```
public static int RoundToIntAwayFrom(this float toRound, int awayFrom)
```

## Parameters

toRound [float](#)

awayFrom [int](#)

## Returns

[int](#)

- `ceil(toRound)` if `toRound >= awayFrom`
- `floor(toRound)` if `toRound < awayFrom`

## RoundToIntTowards(float, int)

Rounds `toRound` to an integer, choosing to round up or down so that it becomes closer (or unchanged) to `towards`.

```
public static int RoundToIntTowards(this float toRound, int towards)
```

Parameters

toRound [float](#)

towards [int](#)

Returns

[int](#)

- `floor(toRound)` if `toRound >= towards`
- `ceil(toRound)` if `toRound < towards`

## RoundToMultipleOf(float, int)

Returns the multiple of `multipleOf` that is closest to `toRound`.

```
public static int RoundToMultipleOf(this float toRound, int multipleOf)
```

Parameters

toRound [float](#)

multipleOf [int](#)

Returns

[int](#)

Exceptions

[ArgumentException](#)

`multipleOf` is 0.

## RoundToMultipleOf(float, float)

Returns the multiple of `multipleOf` that is closest to `toRound`.

```
public static float RoundToMultipleOf(this float toRound, float multipleOf)
```

Parameters

`toRound` [float](#)

`multipleOf` [float](#)

Returns

[float](#)

Exceptions

[ArgumentException](#)

`multipleOf` is 0.

## RoundTowards(float, int)

Rounds `toRound` to an integer, choosing to round up or down so that it becomes closer (or unchanged) to `towards`.

```
public static float RoundTowards(this float toRound, int towards)
```

Parameters

`toRound` [float](#)

`towards` [int](#)

Returns

[float](#)

- `floor(toRound)` if `toRound >= towards`
- `ceil(toRound)` if `toRound < towards`

## Sign(float)

Returns the sign of the given value.

```
public static float Sign(this float x)
```

Parameters

x [float](#)

Returns

[float](#)

- 1 if  $x > 0$
- 0 if  $x = 0$
- -1 if  $x < 0$

Remarks

This is different from Unity's Mathf.Sign(float) since this method defines the sign of 0 to be 0, whereas Mathf.Sign(float) defines it to be 1.

## Square(int)

Computes  $n * n$ .

```
public static float Square(this int n)
```

Parameters

n [int](#)

Returns

[float](#)

## Square(float)

Computes  $n * n$ .

```
public static float Square(this float n)
```

Parameters

$n$  [float](#)

Returns

[float](#)

# Namespace PAC.Patterns

## Classes

### Checkerboard<T>

A repeating pattern of two values where horizontally/vertically adjacent points do not have the same value (unless the two values of the checkerboard are the same).

For example, with r = red, b = blue:

```
r b r b r  
b r b r b  
r b r b r . . .  
b r b r b  
r b r b r
```

### Gradient

Provides a collection of gradient patterns.

#### Gradient.Linear

A gradient that linearly interpolates from the colour of [start](#) to the colour of [end](#) based on how far a point is in the direction of the vector between these two endpoints.

[https://en.wikipedia.org/wiki/Color\\_gradient#Axial\\_gradients](https://en.wikipedia.org/wiki/Color_gradient#Axial_gradients)

#### Gradient.Radial

A gradient that linearly interpolates from the colour of [centre](#) to the colour of [on Circumference](#) based on the distance of a point from the [centre](#).

[https://en.wikipedia.org/wiki/Color\\_gradient#Radial\\_gradients](https://en.wikipedia.org/wiki/Color_gradient#Radial_gradients)

## Interfaces

### IPattern2D<T>

Represents an algorithm to assign a value to each point in the 2D plane.

# Class Checkerboard<T>

Namespace: [PAC.Patterns](#)

A repeating pattern of two values where horizontally/vertically adjacent points do not have the same value (unless the two values of the checkerboard are the same).

For example, with r = red, b = blue:

```
.
.
.
r b r b r
b r b r b
. . . r b r b r . . .
b r b r b
r b r b r
.
.
```

```
public record Checkerboard<T> : IPattern2D<T>, IEquatable<Checkerboard<T>>
```

## Type Parameters

T

### Inheritance

[object](#) ↳ Checkerboard<T>

### Implements

[IPattern2D](#)<T>, [IEquatable](#)<Checkerboard<T>>

### Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

### Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

# Constructors

## Checkerboard(T, T)

```
public Checkerboard(T valueOfOrigin, T otherValue)
```

### Parameters

**valueOfOrigin** T

The value of (0, 0) in the checkerboard.

**otherValue** T

The value of (1, 0) in the checkerboard.

# Properties

## this[IntVector2]

Gets the value of the pattern at the given point.

```
public T this[IntVector2 point] { get; }
```

### Parameters

**point** [IntVector2](#)

### Property Value

T

### Remarks

This value should be completely determined by the state of the object, and computing it should not change the state in a way that will affect any subsequent computations. See [IPattern2D<T>](#) for more details.

# Class Gradient

Namespace: [PAC.Patterns](#)

Provides a collection of gradient patterns.

```
public static class Gradient
```

## Inheritance

[object](#) ← Gradient

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

# Class Gradient.Linear

Namespace: [PAC.Patterns](#)

A gradient that linearly interpolates from the colour of [start](#) to the colour of [end](#) based on how far a point is in the direction of the vector between these two endpoints.

[https://en.wikipedia.org/wiki/Color\\_gradient#Axial\\_gradients](https://en.wikipedia.org/wiki/Color_gradient#Axial_gradients)

```
public record Gradient.Linear : IPattern2D<Color>, IEquatable<Gradient.Linear>
```

## Inheritance

[object](#) ← Gradient.Linear

## Implements

[IPattern2D](#)<Color>, [IEquatable](#)<[Gradient.Linear](#)>

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#),  
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#),  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#),  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#),  
[IPattern2DExtensions.ToTexture\(IPattern2D<Color>, IntRect\)](#)

## Remarks

### In More Detail:

We imagine a (real) line through [start](#) and [end](#). Given a point, we project it perpendicularly onto the line. The colour is then the linear interpolation from the [start](#) colour to the [end](#) colour using the proportion [distance of projected point from start / distance of end from start](#).

Points that project to outside the segment of the line between [start](#) and [end](#) will be given the colour of the closer endpoint. In other words, the colour is clamped between the [start](#) colour and [end](#) colour.

If the coord of [start](#) is the same as the coord of [end](#), then the pattern will be the [start](#) colour everywhere.

# Constructors

`Linear((IntVector2 coord, Color colour), (IntVector2 coord, Color colour))`

See [Gradient.Linear](#) for details.

```
public Linear((IntVector2 coord, Color colour) start, (IntVector2 coord, Color colour) end)
```

## Parameters

`start (IntVector2 coord, Color colour)`

The position and colour of the start of the gradient.

`end (IntVector2 coord, Color colour)`

The position and colour of the end of the gradient.

# Properties

`this[IntVector2]`

Gets the value of the pattern at the given point.

```
public Color this[IntVector2 point] { get; }
```

## Parameters

`point IntVector2`

Property Value

Color

## Remarks

This value should be completely determined by the state of the object, and computing it should not change the state in a way that will affect any subsequent computations. See

[IPattern2D<T>](#) for more details.

## end

The position and colour of the end of the gradient.

```
public (IntVector2 coord, Color colour) end { get; init; }
```

## Property Value

([IntVector2 coord](#), Color [colour](#))

### See Also

[start](#)

## start

The position and colour of the start of the gradient.

```
public (IntVector2 coord, Color colour) start { get; init; }
```

## Property Value

([IntVector2 coord](#), Color [colour](#))

### See Also

[end](#)

# Class Gradient.Radial

Namespace: [PAC.Patterns](#)

A gradient that linearly interpolates from the colour of [centre](#) to the colour of [onCircumference](#) based on the distance of a point from the [centre](#).

[https://en.wikipedia.org/wiki/Color\\_gradient#Radial\\_gradients](https://en.wikipedia.org/wiki/Color_gradient#Radial_gradients)

```
public record Gradient.Radial : IPattern2D<Color>, IEquatable<Gradient.Radial>
```

## Inheritance

[object](#) ← Gradient.Radial

## Implements

[IPattern2D](#)<Color>, [IEquatable](#)<[Gradient.Radial](#)>

## Inherited Members

[object.Equals\(object\)](#), [object.Equals\(object, object\)](#), [object.GetHashCode\(\)](#),  
[object.GetType\(\)](#), [object.MemberwiseClone\(\)](#), [object.ReferenceEquals\(object, object\)](#),  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#),  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#),  
[IPattern2DExtensions.ToTexture\(IPattern2D<Color>, IntRect\)](#)

## Remarks

### In More Detail:

Let  $r = \text{distance}(\text{centre}, \text{onCircumference})$ , and for a given point let  $d = \text{distance}(\text{centre}, \text{point})$ .

- If  $d = 0$ , the point's colour will be the colour of [centre](#).
- If  $d \geq r$ , the point's colour will be the colour of [onCircumference](#).
- If  $0 < d < r$ , the point's colour will be linearly interpolated from the colour of [centre](#) to the colour of [onCircumference](#), using the proportion  $d / r$ .

### Design Note:

If [centre](#) and [onCircumference](#) have the same coord, the [centre](#) will still have its assigned colour and all other points will have the colour of [onCircumference](#). This definition was chosen because I think it feels natural for a user of the gradient tool to always see the [centre](#) have its assigned colour.

## Constructors

`Radial((IntVector2 coord, Color colour), (IntVector2 coord, Color colour))`

See [Gradient.Radial](#) for details.

```
public Radial((IntVector2 coord, Color colour) centre, (IntVector2 coord, Color colour) onCircumference)
```

## Parameters

`centre (IntVector2 coord, Color colour)`

The position and colour of the centre of the circle.

`onCircumference (IntVector2 coord, Color colour)`

The position and colour of some point on the circumference of the circle.

## Properties

`this[IntVector2]`

Gets the value of the pattern at the given point.

```
public Color this[IntVector2 point] { get; }
```

## Parameters

`point IntVector2`

Property Value

## Color

### Remarks

This value should be completely determined by the state of the object, and computing it should not change the state in a way that will affect any subsequent computations. See [IPattern2D<T>](#) for more details.

## centre

The position and colour of the centre of the circle.

```
public (IntVector2 coord, Color colour) centre { get; init; }
```

### Property Value

([IntVector2 coord](#), Color [colour](#))

### See Also

[onCircumference](#)

## onCircumference

The position and colour of some point on the circumference of the circle.

```
public (IntVector2 coord, Color colour) onCircumference { get; init; }
```

### Property Value

([IntVector2 coord](#), Color [colour](#))

### See Also

[centre](#)

# Interface IPattern2D<T>

Namespace: [PAC.Patterns](#)

Represents an algorithm to assign a value to each point in the 2D plane.

```
public interface IPattern2D<out T>
```

## Type Parameters

T

The type of the values assigned to points in the 2D plane.

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Remarks

This value should be completely determined by the state of the object, and computing it should not change the state in a way that will affect any subsequent computations. This is so the value is invariant across multiple calls, but allows 'settings' of the pattern to be changed.

## Intended Usage:

```
class Pattern : IPattern2D<int>
{
    public int valueOfZero;

    public Pattern(int valueOfZero)
    {
        this.valueOfZero = valueOfZero;
    }

    // Assigns valueOfZero to (0, 0) and assigns 0 to every other point
    public int this[IntVector2 point] => point == IntVector2.zero ? valueOfZero : 0;
}
```

Here the value of a point is completely determined by `valueOfZero` and computing a value does not change anything about the object. The user is allowed to change `valueOfZero`, which acts as a 'setting' for the pattern.

### Not Intended Usage:

```
class Pattern : IPattern2D<int>
{
    private int value = 0;

    public int this[IntVector2 point]
    {
        value += 1; // Changes the object's state
        return value;
    }
}
```

This is not intended usage because computing a value modifies the state of the object so that computing the value of, say, `(0, 0)` twice will give two different values without having changed any 'settings' of the pattern.

### Note:

Changes to state that don't affect the values are allowed. So, for example, caching values to avoid recomputing them is allowed.

## Properties

### this[IntVector2]

Gets the value of the pattern at the given point.

```
T this[IntVector2 point] { get; }
```

#### Parameters

`point` [IntVector2](#)

#### Property Value

T

## Remarks

This value should be completely determined by the state of the object, and computing it should not change the state in a way that will affect any subsequent computations. See [`IPattern2D<T>`](#) for more details.

# Namespace PAC.Screen

## Classes

[MaximiseWindow](#)

[ScreenInfo](#)

# Class MaximiseWindow

Namespace: [PAC.Screen](#)

```
public class MaximiseWindow : MonoBehaviour
```

## Inheritance

[object](#) ← MaximiseWindow

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### ShowWindowAsync(int, int)

```
public static extern bool ShowWindowAsync(int hWnd, int nCmdShow)
```

#### Parameters

hWnd [int](#)

nCmdShow [int](#)

#### Returns

[bool](#)

# Class ScreenInfo

Namespace: [PAC.Screen](#)

```
public static class ScreenInfo
```

## Inheritance

[object](#) ← ScreenInfo

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Properties

### screenWorldHeight

Returns the height of the screen in world coords.

```
public static float screenWorldHeight { get; }
```

Property Value

[float](#)

### screenWorldWidth

Returns the width of the screen in world coords.

```
public static float screenWorldWidth { get; }
```

Property Value

[float](#)

# Methods

## GetScreenPixelColour(IntVector2)

Returns the colour of the screen pixel at the given coords. Best results when called after end of frame, using 'yield return new WaitForEndOfFrame()'.

```
public static Color GetScreenPixelColour(IntVector2 coords)
```

Parameters

coords [IntVector2](#)

Returns

Color

## GetScreenPixelColour(int, int)

Returns the colour of the screen pixel at coords (x, y). Best results when called after end of frame, using 'yield return new WaitForEndOfFrame()'.

```
public static Color GetScreenPixelColour(int x, int y)
```

Parameters

x [int](#)

y [int](#)

Returns

Color

# Namespace PAC.Serialization

## Classes

[LayerSerializationSurrogate](#)

[Texture2DSerializationSurrogate](#)

# Class LayerSerializationSurrogate

Namespace: [PAC.Serialization](#)

```
[Obsolete]
public class LayerSerializationSurrogate : ISerializationSurrogate
```

## Inheritance

[object](#) ← LayerSerializationSurrogate

## Implements

[ISerializationSurrogate](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### GetObjectData(object, SerializationInfo, StreamingContext)

Populates the provided [SerializationInfo](#) with the data needed to serialize the object.

```
public void GetObjectData(object obj, SerializationInfo info,
StreamingContext context)
```

#### Parameters

**obj** [object](#)

The object to serialize.

**info** [SerializationInfo](#)

The [SerializationInfo](#) to populate with data.

**context** [StreamingContext](#)

The destination (see [StreamingContext](#)) for this serialization.

## Exceptions

[SecurityException](#)

The caller does not have the required permission.

# SetObjectData(object, SerializationInfo, StreamingContext, ISurrogateSelector)

Populates the object using the information in the [SerializationInfo](#).

```
public object SetObjectData(object obj, SerializationInfo info, StreamingContext context, ISurrogateSelector selector)
```

## Parameters

**obj** [object](#)

The object to populate.

**info** [SerializationInfo](#)

The information to populate the object.

**context** [StreamingContext](#)

The source from which the object is deserialized.

**selector** [ISurrogateSelector](#)

The surrogate selector where the search for a compatible surrogate begins.

## Returns

[object](#)

The populated deserialized object.

## Exceptions

### [SecurityException](#)

The caller does not have the required permission.

# Class Texture2DSerializationSurrogate

Namespace: [PAC.Serialization](#)

```
[Obsolete]
public class Texture2DSerializationSurrogate : ISerializationSurrogate
```

## Inheritance

[object](#) ← Texture2DSerializationSurrogate

## Implements

[ISerializationSurrogate](#)

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### GetObjectData(object, SerializationInfo, StreamingContext)

Populates the provided [SerializationInfo](#) with the data needed to serialize the object.

```
public void GetObjectData(object obj, SerializationInfo info,
StreamingContext context)
```

#### Parameters

**obj** [object](#)

The object to serialize.

**info** [SerializationInfo](#)

The [SerializationInfo](#) to populate with data.

**context** [StreamingContext](#)

The destination (see [StreamingContext](#)) for this serialization.

## Exceptions

[SecurityException](#)

The caller does not have the required permission.

# SetObjectData(object, SerializationInfo, StreamingContext, ISurrogateSelector)

Populates the object using the information in the [SerializationInfo](#).

```
public object SetObjectData(object obj, SerializationInfo info, StreamingContext context, ISurrogateSelector selector)
```

## Parameters

**obj** [object](#)

The object to populate.

**info** [SerializationInfo](#)

The information to populate the object.

**context** [StreamingContext](#)

The source from which the object is deserialized.

**selector** [ISurrogateSelector](#)

The surrogate selector where the search for a compatible surrogate begins.

## Returns

[object](#)

The populated deserialized object.

## Exceptions

### [SecurityException](#)

The caller does not have the required permission.

# Namespace PAC.Shaders

## Classes

[HueSaturationBoxShader](#)

[LightnessSliderShader](#)

[Outline](#)

[RainbowOutline](#)

# Class HueSaturationBoxShader

Namespace: [PAC.Shaders](#)

```
public class HueSaturationBoxShader : MonoBehaviour
```

## Inheritance

[object](#) ← HueSaturationBoxShader

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### hueSaturationBoxMaterial

```
public Material hueSaturationBoxMaterial
```

Field Value

Material

# Class LightnessSliderShader

Namespace: [PAC.Shaders](#)

```
public class LightnessSliderShader : MonoBehaviour
```

## Inheritance

[object](#) ← LightnessSliderShader

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### lightnessSliderMaterial

```
public Material lightnessSliderMaterial
```

#### Field Value

Material

# Class Outline

Namespace: [PAC.Shaders](#)

```
public class Outline : MonoBehaviour
```

## Inheritance

[object](#) ← Outline

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### outlineMaterial

```
public Material outlineMaterial
```

#### Field Value

Material

## Properties

### colour

```
public Color colour { get; set; }
```

#### Property Value

Color

### keepExistingTexture

```
public bool keepExistingTexture { get; set; }
```

Property Value

[bool](#) ↗

## outlineEnabled

```
public bool outlineEnabled { get; set; }
```

Property Value

[bool](#) ↗

## thickness

```
public float thickness { get; set; }
```

Property Value

[float](#) ↗

# Class RainbowOutline

Namespace: [PAC.Shaders](#)

```
public class RainbowOutline : MonoBehaviour
```

## Inheritance

[object](#) ← RainbowOutline

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### rainbowOutlineMaterial

```
public Material rainbowOutlineMaterial
```

Field Value

Material

## Properties

### keepExistingTexture

```
public bool keepExistingTexture { get; set; }
```

Property Value

[bool](#)

### outlineEnabled

```
public bool outlineEnabled { get; set; }
```

Property Value

[bool](#) ↗

## thickness

```
public float thickness { get; set; }
```

Property Value

[float](#) ↗

# Namespace PAC.Shapes

## Classes

### Diamond

A pixel art diamond shape. For example,

```
#  
#  
#   #  
#   #  
#       #  
#       #  
#   #  
#   #  
#  
#
```

### Ellipse

A pixel art ellipse shape. For example,

```
# # # #  
#           #  
#           #  
#           #  
#           #  
# # # #
```

### IsometricCuboid

An isometric pixel art cuboid shape. For example,

```
    # #  
# #     # #  
# #           # #  
# # #           # #  
#       # #     # #  
#           # #     #  
# #           #     #  
# #     #     # #  
# #   # # # #  
# #
```

### IsometricRectangle

An isometric pixel art rectangle shape. For example,

```
# #
# #      # #
# #          # #
# #          # #
# #      # #
# #
```

## Line

A pixel art straight line. For example,

```
#  
# #  
#  
# #  
#
```

## Path

A sequence of connected [Lines](#). For example,

```
end of line 1  
start of line 2    # < end of line 3  
      v          #  
      # #        #  
      #      # #    # < start of line 3  
      #          # #  
      #          ^  
      ^          end of line 2  
start of line 1
```

## [Rectangle](#)

A pixel art rectangle shape. For example,

```
# # # #
#      #
# # # #
```

## [RightTriangle](#)

A pixel art right-angled triangle shape. For example,

```
    # #
# #   #
#       #
# #       #
#
# # # # # # # #
```

## Enums

### [RightTriangle.RightAngleLocation](#)

Which corner of the [boundingRect](#) the right angle is in.

# Class Diamond

Namespace: [PAC.Shapes](#)

A pixel art diamond shape. For example,

```
#  
#  
#  #  
#  #  
#      #  
#      #  
#  #  
#  #  
#  
#
```

  

```
public class Diamond : I2DShape<Diamond>, IFillableShape,  
ITransformableShape<Diamond>, ITranslatableShape<Diamond>, IFlippableShape<Diamond>,  
IRotatableShape<Diamond>, IDeepCopyableShape<Diamond>, IShape,  
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,  
IEnumerable<IntVector2>, IEnumerable, IEquatable<Diamond>
```

## Inheritance

[object](#) ← Diamond

## Implements

[I2DShape<Diamond>](#), [IFillableShape](#), [ITransformableShape<Diamond>](#),  
[ITranslatableShape<Diamond>](#), [IFlippableShape<Diamond>](#), [IRotatableShape<Diamond>](#),  
[IDeepCopyableShape<Diamond>](#), [IShape](#), [IReadOnlyContains<IntVector2>](#),  
[IReadOnlyCollection<IntVector2>](#), [IEnumerable<IntVector2>](#), [IEnumerable](#),  
[IEquatable<Diamond>](#)

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,

[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(I Shape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(I Shape, Color, IntRect\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToStringPrettyString<T>\(IEnumerable<T>\)](#) ,

[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Remarks

### Shape:

If the border can be drawn using perfect [Lines](#), it will be. (See [Line](#) for a definition of 'perfect').

If the width or height of the [boundingRect](#) is  $\leq 2$ , it will look like a rectangle.

### Enumerator:

The enumerator does not repeat any points.

## Constructors

### Diamond(IntRect, bool)

Creates the largest [Diamond](#) that can fit in the given rect.

```
public Diamond(IntRect boundingRect, bool filled)
```

## Parameters

[boundingRect](#) [IntRect](#)

See [boundingRect](#).

[filled](#) [bool](#) ↗

See [filled](#).

# Properties

## Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

## Property Value

[int](#)

The number of elements in the collection.

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; set; }
```

## Property Value

[IntRect](#)

## filled

Whether the shape has its inside filled-in, or whether it's just the border.

```
public bool filled { get; set; }
```

## Property Value

[bool](#)

## isSquare

Whether the [Diamond](#) is a square (at a 45-degree angle).

```
public bool isSquare { get; }
```

Property Value

[bool](#) ↗

Remarks

This is equivalent to its [boundingRect](#) being a square.

## Methods

### Contains(IntVector2)

```
public bool Contains(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#) ↗

### DeepCopy()

Returns a deep copy of the shape.

```
public Diamond DeepCopy()
```

Returns

[Diamond](#)

### Equals(Diamond)

See [operator ==\(Diamond, Diamond\)](#).

```
public bool Equals(Diamond other)
```

Parameters

other [Diamond](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(Diamond\)](#).

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public Diamond Flip(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

Returns

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumrator<IntVector2> GetEnumerator()
```

Returns

[IEnumrator](#)<IntVector2>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## Rotate(RotationAngle)

Returns a deep copy of the shape rotated by the given angle.

```
public Diamond Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

## [Diamond](#)

### **See Also**

operator `-(!RotatableShape<T>)`

## [ToString\(\)](#)

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## [Translate\(IntVector2\)](#)

Returns a deep copy of the shape translated by the given vector.

```
public Diamond Translate(IntVector2 translation)
```

Parameters

`translation` [IntVector2](#)

Returns

[Diamond](#)

### **See Also**

operator `+(ITranslatableShape<T>, IntVector2)`,  
operator `±(IntVector2, ITranslatableShape<T>)`,  
operator `-(!TranslatableShape<T>, IntVector2)`

## Operators

## operator +(IntVector2, Diamond)

Returns a deep copy of the [Diamond](#) translated by the given vector.

```
public static Diamond operator +(IntVector2 translation, Diamond diamond)
```

Parameters

translation [IntVector2](#)

diamond [Diamond](#)

Returns

[Diamond](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator +(Diamond, IntVector2)

Returns a deep copy of the [Diamond](#) translated by the given vector.

```
public static Diamond operator +(Diamond diamond, IntVector2 translation)
```

Parameters

diamond [Diamond](#)

translation [IntVector2](#)

Returns

[Diamond](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator ==(Diamond, Diamond)

Whether the two [Diamond](#)s have the same shape, and the same value for [filled](#).

```
public static bool operator ==(Diamond a, Diamond b)
```

Parameters

a [Diamond](#)

b [Diamond](#)

Returns

[bool](#)

## operator !=(Diamond, Diamond)

See [operator ==\(Diamond, Diamond\)](#).

```
public static bool operator !=(Diamond a, Diamond b)
```

Parameters

a [Diamond](#)

b [Diamond](#)

Returns

[bool](#)

## operator -(Diamond, IntVector2)

Returns a deep copy of the [Diamond](#) translated by the given vector.

```
public static Diamond operator -(Diamond diamond, IntVector2 translation)
```

Parameters

[diamond](#) [Diamond](#)

[translation](#) [IntVector2](#)

Returns

[Diamond](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator -(Diamond)

Returns a deep copy of the [Diamond](#) rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static Diamond operator -(Diamond diamond)
```

Parameters

[diamond](#) [Diamond](#)

Returns

[Diamond](#)

**See Also**

[Rotate\(RotationAngle\)](#), [Flip\(FlipAxis\)](#)

# Class Ellipse

Namespace: [PAC.Shapes](#)

A pixel art ellipse shape. For example,

```
# # # #
#       #
#       #
#       #
#       #
# # # #
```

```
public class Ellipse : I2DShape<Ellipse>, IFillableShape,
ITransformableShape<Ellipse>, ITranslatableShape<Ellipse>, IFlippableShape<Ellipse>,
IRotatableShape<Ellipse>, IDeepCopyableShape<Ellipse>, IShape,
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,
IEnumerable<IntVector2>, IEnumerable, IEquatable<Ellipse>
```

## Inheritance

[object](#) ↗ ← Ellipse

## Implements

[I2DShape<Ellipse>](#), [IFillableShape](#), [ITransformableShape<Ellipse>](#),  
[ITranslatableShape<Ellipse>](#), [IFlippableShape<Ellipse>](#), [IRotatableShape<Ellipse>](#),  
[IDeepCopyableShape<Ellipse>](#), [IShape](#), [IReadOnlyContains<IntVector2>](#),  
[IReadOnlyCollection<IntVector2>](#), [IEnumerable<IntVector2>](#), [IEnumerable](#),  
[IEquatable<Ellipse>](#)

## Inherited Members

[object.Equals\(object, object\)](#) ↗ , [object.GetType\(\)](#) ↗ , [object.MemberwiseClone\(\)](#) ↗ ,  
[object.ReferenceEquals\(object, object\)](#) ↗

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,

[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(I Shape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(I Shape, Color, IntRect\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,

[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>,  
IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

### Shape:

If the width or height of the [boundingRect](#) is  $\leq 2$ , it will look like a rectangle.

### Enumerator:

The enumerator does not repeat any points.

## Constructors

### Ellipse(IntRect, bool)

Creates the largest [Ellipse](#) that can fit in the given rect.

```
public Ellipse(IntRect boundingRect, bool filled)
```

#### Parameters

[boundingRect](#) [IntRect](#)

See [boundingRect](#).

[filled](#) [bool](#) ↗

See [filled](#).

## Properties

### Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

Property Value

[int](#)

The number of elements in the collection.

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; set; }
```

Property Value

[IntRect](#)

## filled

Whether the shape has its inside filled-in, or whether it's just the border.

```
public bool filled { get; set; }
```

Property Value

[bool](#)

## isCircle

Whether the [Ellipse](#) is a circle.

```
public bool isCircle { get; }
```

Property Value

[bool](#) ↗

Remarks

This is equivalent to its [boundingRect](#) being a square.

## Methods

### Contains(IntVector2)

`public bool Contains(IntVector2 point)`

Parameters

`point` [IntVector2](#)

Returns

[bool](#) ↗

### DeepCopy()

Returns a deep copy of the shape.

`public Ellipse DeepCopy()`

Returns

[Ellipse](#)

### Equals(Ellipse)

See [operator ==\(Ellipse, Ellipse\)](#).

```
public bool Equals(Ellipse other)
```

Parameters

other [Ellipse](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(Ellipse\)](#).

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public Ellipse Flip(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

Returns

[Ellipse](#)

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumrator<IntVector2> GetEnumerator()
```

Returns

[IEnumrator](#)<IntVector2>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## Rotate(RotationAngle)

Returns a deep copy of the shape rotated by the given angle.

```
public Ellipse Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

## [Ellipse](#)

### **See Also**

operator `-`([IRotatableShape](#)<T>)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public Ellipse Translate(IntVector2 translation)
```

Parameters

`translation` [IntVector2](#)

Returns

[Ellipse](#)

### **See Also**

operator `+`([ITranslatableShape](#)<T>, [IntVector2](#)),

operator `±`([IntVector2](#), [ITranslatableShape](#)<T>),

operator `-`([ITranslatableShape](#)<T>, [IntVector2](#))

## Operators

## operator +(IntVector2, Ellipse)

Returns a deep copy of the [Ellipse](#) translated by the given vector.

```
public static Ellipse operator +(IntVector2 translation, Ellipse ellipse)
```

Parameters

`translation` [IntVector2](#)

`ellipse` [Ellipse](#)

Returns

[Ellipse](#)

### See Also

[Translate\(IntVector2\)](#)

## operator +(Ellipse, IntVector2)

Returns a deep copy of the [Ellipse](#) translated by the given vector.

```
public static Ellipse operator +(Ellipse ellipse, IntVector2 translation)
```

Parameters

`ellipse` [Ellipse](#)

`translation` [IntVector2](#)

Returns

[Ellipse](#)

### See Also

[Translate\(IntVector2\)](#)

## operator ==(Ellipse, Ellipse)

Whether the two [Ellipses](#) have the same shape, and the same value for [filled](#).

```
public static bool operator ==(Ellipse a, Ellipse b)
```

Parameters

a [Ellipse](#)

b [Ellipse](#)

Returns

[bool](#)

## operator !=(Ellipse, Ellipse)

See [operator ==\(Ellipse, Ellipse\)](#).

```
public static bool operator !=(Ellipse a, Ellipse b)
```

Parameters

a [Ellipse](#)

b [Ellipse](#)

Returns

[bool](#)

## operator -(Ellipse, IntVector2)

Returns a deep copy of the [Ellipse](#) translated by the given vector.

```
public static Ellipse operator -(Ellipse ellipse, IntVector2 translation)
```

Parameters

`ellipse` [Ellipse](#)

`translation` [IntVector2](#)

Returns

[Ellipse](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator -(Ellipse)

Returns a deep copy of the [Ellipse](#) rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static Ellipse operator -(Ellipse ellipse)
```

Parameters

`ellipse` [Ellipse](#)

Returns

[Ellipse](#)

**See Also**

[Rotate\(RotationAngle\)](#), [Flip\(FlipAxis\)](#)

# Class IsometricCuboid

Namespace: [PAC.Shapes](#)

An isometric pixel art cuboid shape. For example,

```
# #
# #      # #
# #          # #
#   # #      # #
#       # #      # #
#           # #      #
# #          #      #
# #      #      # #
# #      # # # #
# #
```

```
public class IsometricCuboid : IIsoMetricShape<IsometricCuboid>, IFillableShape,
ITranslatableShape<IsometricCuboid>, IFlippableShape<IsometricCuboid>,
IDeepCopyableShape<IsometricCuboid>, IShape, IReadOnlyContains<IntVector2>,
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>,
IEnumerable, IEquatable<IsometricCuboid>
```

## Inheritance

[object](#) ← IsometricCuboid

## Implements

[IIsoMetricShape<IsometricCuboid>](#), [IFillableShape](#), [ITranslatableShape<IsometricCuboid>](#),  
[IFlippableShape<IsometricCuboid>](#), [IDeepCopyableShape<IsometricCuboid>](#), [IShape](#),  
[IReadOnlyContains<IntVector2>](#), [IReadOnlyCollection<IntVector2>](#),  
[IEnumerable<IntVector2>](#), [IEquatable<IsometricCuboid>](#)

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,

[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#),  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#),  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#),  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#),  
[IShapeExtensions.ToTexture\(IShape, Color\)](#),  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#),  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#),  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#),  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#),  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#),

[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TTThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TTThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

### Shape:

The top/bottom faces are identical [IsometricRectangle](#)s.

If you imagine a real solid 3D cuboid, then 3 of the 12 edges are not visible in the isometric view as they are at the back of the cuboid. Whether the [IsometricCuboid](#) includes these edges or not is determined by [includeBackEdges](#). For example, the example [Isometric Cuboid](#) diagram above has [includeBackEdges](#) set to [false](#). Setting it to [true](#) would look like:

```
# #
# # #   # #
# #     #       # #
# # # #           # #
#       # #       # #   #
# # #     # #       #
# #           # # #   #
# #     #       # #
# #   # # # #
# #
```

### Enumerator:

The enumerator may repeat some points.

## Constructors

### IsometricCuboid(IsometricRectangle, int, bool, bool)

Creates an [IsometricCuboid](#) with the given [IsometricRectangle](#) as either:

- The bottom face, if `height >= 0`
- The top face, if `height < 0`

```
public IsometricCuboid(IsometricRectangle topOrBottomFace, int height, bool filled,  
bool includeBackEdges)
```

## Parameters

`topOrBottomFace` [IsometricRectangle](#)

`height` [int](#)

See [height](#).

`filled` [bool](#)

See [filled](#).

`includeBackEdges` [bool](#)

See [includeBackEdges](#).

## Remarks

The [filled](#) property of the [IsometricRectangle](#) is ignored, and the parameter `filled` is used instead.

The [IsometricRectangle](#) is not deep-copied, so changes to it will affect the [IsometricCuboid](#) as well.

# Properties

## Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

## Property Value

[int](#)

The number of elements in the collection.

## bottomFace

The bottom face of the [IsometricCuboid](#).

```
public IsometricRectangle bottomFace { get; }
```

Property Value

[IsometricRectangle](#)

Remarks

The [filled](#) property of this is ignored, and the [filled](#) property of the [IsometricCuboid](#) is used instead.

### See Also

[topFace](#)

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; }
```

Property Value

[IntRect](#)

## filled

Whether the shape has its inside filled-in, or whether it's just the border.

```
public bool filled { get; set; }
```

Property Value

[bool](#)

## height

The height of the cuboid if you imagined it in 3D space. In other words, how much [bottom Face](#) is translated upward to form [topFace](#).

```
public int height { get; set; }
```

Property Value

[int](#)

## includeBackEdges

Whether or not to show the 3 edges at the back of the cuboid. See [IsometricCuboid](#) for more details.

```
public bool includeBackEdges { get; set; }
```

Property Value

[bool](#)

## Remarks

This has no effect if [filled](#) is [true](#).

## topFace

The top face of the [IsometricCuboid](#).

```
public IsometricRectangle topFace { get; }
```

Property Value

[IsometricRectangle](#)

## Remarks

The [filled](#) property of this is ignored, and the [filled](#) property of the [IsometricCuboid](#) is used instead.

## See Also

[bottomFace](#)

## Methods

### Contains(IntVector2)

```
public bool Contains(IntVector2 point)
```

#### Parameters

point [IntVector2](#)

#### Returns

[bool](#) ↗

### DeepCopy()

Returns a deep copy of the shape.

```
public IsometricCuboid DeepCopy()
```

#### Returns

[IsometricCuboid](#)

### Equals(IsometricCuboid)

See [operator ==\(IsometricCuboid, IsometricCuboid\)](#).

```
public bool Equals(IsometricCuboid other)
```

Parameters

**other** [IsometricCuboid](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(IsometricCuboid\)](#).

```
public override bool Equals(object obj)
```

Parameters

**obj** [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public IsometricCuboid Flip(FlipAxis axis)
```

Parameters

**axis** [FlipAxis](#)

Returns

[IsometricCuboid](#)

## Remarks

Can only be flipped across the vertical axis (or none axis).

## Exceptions

### [ArgumentException](#)

`axis` is an invalid axis.

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumarator<IntVector2> GetEnumerator()
```

## Returns

### [IEnumarator](#)<IntVector2>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

## Returns

### [int](#)

A hash code for the current object.

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public IsometricCuboid Translate(IntVector2 translation)
```

Parameters

translation [IntVector2](#)

Returns

[IsometricCuboid](#)

### See Also

operator +(ITranslatableShape<T>, IntVector2),  
operator +(IntVector2, ITranslatableShape<T>),  
operator -(ITranslatableShape<T>, IntVector2)

## Operators

### operator +(IntVector2, IsometricCuboid)

Returns a deep copy of the [IsometricCuboid](#) translated by the given vector.

```
public static IsometricCuboid operator +(IntVector2 translation,  
IsometricCuboid isometricCuboid)
```

Parameters

`translation` [IntVector2](#)

`isometricCuboid` [IsometricCuboid](#)

Returns

[IsometricCuboid](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator +(IsometricCuboid, IntVector2)

Returns a deep copy of the [IsometricCuboid](#) translated by the given vector.

```
public static IsometricCuboid operator +(IsometricCuboid isometricCuboid,  
IntVector2 translation)
```

Parameters

`isometricCuboid` [IsometricCuboid](#)

`translation` [IntVector2](#)

Returns

[IsometricCuboid](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator ==(IsometricCuboid, IsometricCuboid)

Whether the two [IsometricCuboids](#) have the same [bottomFace](#) and the same values for [height](#), [filled](#) and [includeBackEdges](#).

```
public static bool operator ==(IsometricCuboid a, IsometricCuboid b)
```

Parameters

a [IsometricCuboid](#)

b [IsometricCuboid](#)

Returns

[bool](#)

Remarks

Note that if two [IsometricCuboids](#) have identical values except their [heights](#) differ in sign then they will look identical but will not be considered ==.

## operator !=(IsometricCuboid, IsometricCuboid)

See [operator ==\(IsometricCuboid, IsometricCuboid\)](#).

```
public static bool operator !=(IsometricCuboid a, IsometricCuboid b)
```

Parameters

a [IsometricCuboid](#)

b [IsometricCuboid](#)

Returns

[bool](#)

## operator -(IsometricCuboid, IntVector2)

Returns a deep copy of the [IsometricCuboid](#) translated by the given vector.

```
public static IsometricCuboid operator -(IsometricCuboid isometricCuboid,  
IntVector2 translation)
```

Parameters

isometricCuboid [IsometricCuboid](#)

translation [IntVector2](#)

Returns

[IsometricCuboid](#)

**See Also**

[Translate\(IntVector2\)](#).

# Class IsometricRectangle

Namespace: [PAC.Shapes](#)

An isometric pixel art rectangle shape. For example,

```
# #
# #      # #
# #          # #
# #          # #
# #      # #
# #
```

```
public class IsometricRectangle : IIsometricShape<IsometricRectangle>,
IFillableShape, ITranslatableShape<IsometricRectangle>,
IFlippableShape<IsometricRectangle>, IDeepCopyableShape<IsometricRectangle>, IShape,
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,
IEnumerable<IntVector2>, IEnumerable, IEquatable<IsometricRectangle>
```

## Inheritance

[object](#) ← IsometricRectangle

## Implements

[IIsometricShape<IsometricRectangle>](#), [IFillableShape](#),  
[ITranslatableShape<IsometricRectangle>](#), [IFlippableShape<IsometricRectangle>](#),  
[IDeepCopyableShape<IsometricRectangle>](#), [IShape](#), [IReadOnlyContains<IntVector2>](#),  
[IReadOnlyCollection<IntVector2>](#), [IEnumerable<IntVector2>](#), [IEnumerable](#),  
[IEquatable<IsometricRectangle>](#)

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,

[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,

[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

### Shape:

The shape is determined by [startCorner](#) and [endCorner](#), which will be opposite corners of the rectangle. The lines will always be perfect in blocks of 2, except possibly at the left/right/top/bottom blocks of the shape:

- The top/bottom will be a horizontal block of 2 or 3. They will match.
- The left/bottom will either be a single point, a horizontal block of 2 or a vertical block of 2.
  - If one is in the vertical case, so will be the other.
  - In the other two cases, they may not match, but if [endCorner](#) is in a horizontal block of 2, then so is [startCorner](#).

### Enumerator:

The enumerator does not repeat any points.

## Constructors

### IsometricRectangle(IntVector2, IntVector2, bool)

See [IsometricRectangle](#) for details.

```
public IsometricRectangle(IntVector2 startCorner, IntVector2 endCorner, bool filled)
```

## Parameters

### startCorner [IntVector2](#)

See [startCorner](#).

`endCorner` [IntVector2](#)

See [endCorner](#).

`filled` [bool](#) ↗

See [filled](#).

## Fields

### endCorner

The point the mouse has dragged to.

`public IntVector2 endCorner`

Field Value

[IntVector2](#)

Remarks

This will always be one of [leftCorner](#), [rightCorner](#), [bottomCorner](#) or [topCorner](#).

### See Also

[startCorner](#), [leftCorner](#), [rightCorner](#), [bottomCorner](#), [topCorner](#)

### startCorner

The point the mouse started dragging from.

`public IntVector2 startCorner`

Field Value

[IntVector2](#)

Remarks

This will always be one of [leftCorner](#), [rightCorner](#), [bottomCorner](#) or [topCorner](#).

## See Also

[endCorner](#), [leftCorner](#), [rightCorner](#), [bottomCorner](#), [topCorner](#)

# Properties

## Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

### Property Value

[int](#)

The number of elements in the collection.

## border

The outline of the [IsometricRectangle](#).

```
public Path border { get; }
```

### Property Value

[Path](#)

## Remarks

This is the concatenation of [lowerBorder](#) and [upperBorder](#).

## bottomCorner

One of the bottommost points of the [IsometricRectangle](#).

```
public IntVector2 bottomCorner { get; set; }
```

Property Value

[IntVector2](#)

**See Also**

[startCorner](#), [endCorner](#), [leftCorner](#), [rightCorner](#), [topCorner](#)

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; }
```

Property Value

[IntRect](#)

## filled

Whether the shape has its inside filled-in, or whether it's just the border.

```
public bool filled { get; set; }
```

Property Value

[bool](#)

## isIsometricSquare

Whether the sides lengths of the [IsometricRectangle](#) are all equal.

```
public bool isIsometricSquare { get; }
```

Property Value

[bool](#)

## leftCorner

(One of) the leftmost point(s) of the [IsometricRectangle](#).

```
public IntVector2 leftCorner { get; set; }
```

Property Value

[IntVector2](#)

### See Also

[startCorner](#), [endCorner](#), [rightCorner](#), [bottomCorner](#), [topCorner](#)

## lowerBorder

The lower edges of the [border](#), i.e. ([leftCorner](#), [bottomCorner](#)) and ([bottomCorner](#), [rightCorner](#)) - not necessarily in this direction.

```
public Path lowerBorder { get; }
```

Property Value

[Path](#)

### Remarks

This is precisely the set of points in [border](#) that have the lowest y coord among those with the same x coord. (Note there's at most one other point with the same x coord.)

## rightCorner

(One of) the rightmost point(s) of the [IsometricRectangle](#).

```
public IntVector2 rightCorner { get; set; }
```

Property Value

[IntVector2](#)

## See Also

[startCorner](#), [endCorner](#), [leftCorner](#), [bottomCorner](#), [topCorner](#)

## topCorner

One of the topmost points of the [IsometricRectangle](#).

```
public IntVector2 topCorner { get; set; }
```

Property Value

[IntVector2](#)

## See Also

[startCorner](#), [endCorner](#), [leftCorner](#), [rightCorner](#), [bottomCorner](#)

## upperBorder

The upper edges of the [border](#), i.e. ([leftCorner](#), [topCorner](#)) and ([topCorner](#), [rightCorner](#)) - not necessarily in this direction.

```
public Path upperBorder { get; }
```

Property Value

[Path](#)

## Remarks

This is precisely the set of points in [border](#) that have the highest y coord among those with the same x coord. (Note there's at most one other point with the same x coord.)

## Methods

[Contains\(IntVector2\)](#)

```
public bool Contains(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#)

## DeepCopy()

Returns a deep copy of the shape.

```
public IsometricRectangle DeepCopy()
```

Returns

[IsometricRectangle](#)

## Equals(IsometricRectangle)

See [operator ==\(IsometricRectangle, IsometricRectangle\).](#)

```
public bool Equals(IsometricRectangle other)
```

Parameters

other [IsometricRectangle](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(IsometricRectangle\)](#).

```
public override bool Equals(object obj)
```

Parameters

`obj` [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public IsometricRectangle Flip(FlipAxis axis)
```

Parameters

`axis` [FlipAxis](#)

Returns

[IsometricRectangle](#)

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<IntVector2> GetEnumerator()
```

Returns

[IEnumerator](#)<[IntVector2](#)>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public IsometricRectangle Translate(IntVector2 translation)
```

Parameters

[translation](#) [IntVector2](#)

Returns

[IsometricRectangle](#)

## See Also

operator [+\(ITranslatableShape<T>, IntVector2\)](#),  
operator [+\(IntVector2, ITranslatableShape<T>\)](#),  
operator [-\(ITranslatableShape<T>, IntVector2\)](#)

# Operators

## operator +(IntVector2, IsometricRectangle)

Returns a deep copy of the [IsometricRectangle](#) translated by the given vector.

```
public static IsometricRectangle operator +(IntVector2 translation,  
IsometricRectangle isometricRectangle)
```

### Parameters

translation [IntVector2](#)

isometricRectangle [IsometricRectangle](#)

### Returns

[IsometricRectangle](#)

## See Also

[Translate\(IntVector2\)](#).

## operator +(IsometricRectangle, IntVector2)

Returns a deep copy of the [IsometricRectangle](#) translated by the given vector.

```
public static IsometricRectangle operator +(IsometricRectangle isometricRectangle,  
IntVector2 translation)
```

### Parameters

isometricRectangle [IsometricRectangle](#)

translation [IntVector2](#)

Returns

[IsometricRectangle](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator ==(IsometricRectangle, IsometricRectangle)

Whether the two [IsometricRectangle](#)s have the same values for [startCorner](#), [endCorner](#) and [filled](#).

```
public static bool operator ==(IsometricRectangle a, IsometricRectangle b)
```

Parameters

a [IsometricRectangle](#)

b [IsometricRectangle](#)

Returns

[bool](#) ↗

## operator !=(IsometricRectangle, IsometricRectangle)

See [operator ==\(IsometricRectangle, IsometricRectangle\)](#).

```
public static bool operator !=(IsometricRectangle a, IsometricRectangle b)
```

Parameters

a [IsometricRectangle](#)

b [IsometricRectangle](#)

Returns

## operator -(IsometricRectangle, IntVector2)

Returns a deep copy of the [IsometricRectangle](#) translated by the given vector.

```
public static IsometricRectangle operator -(IsometricRectangle isometricRectangle,  
IntVector2 translation)
```

Parameters

isometricRectangle [IsometricRectangle](#)

translation [IntVector2](#)

Returns

[IsometricRectangle](#)

### See Also

[Translate\(IntVector2\)](#)

## operator -(IsometricRectangle)

Returns a deep copy of the [IsometricRectangle](#) rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static IsometricRectangle operator -(IsometricRectangle isometricRectangle)
```

Parameters

isometricRectangle [IsometricRectangle](#)

Returns

[IsometricRectangle](#)

### See Also

[Flip\(FlipAxis\)](#)

# Class Line

Namespace: [PAC.Shapes](#)

A pixel art straight line. For example,

```
#  
# #  
#  
# #  
#  
  
public class Line : I1DShape<Line>, ITransformableShape<Line>,  
ITranslatableShape<Line>, IFlippableShape<Line>, IRotatableShape<Line>,  
IDeepCopyableShape<Line>, IShape, IReadOnlyContains<IntVector2>,  
IReadOnlyList<IntVector2>, IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>,  
IEnumerable, IEquatable<Line>
```

## Inheritance

[object](#) ↗ ← Line

## Implements

[I1DShape<Line>](#), [ITransformableShape<Line>](#), [ITranslatableShape<Line>](#),  
[IFlippableShape<Line>](#), [IRotatableShape<Line>](#), [IDeepCopyableShape<Line>](#), [IShape](#),  
[IReadOnlyContains<IntVector2>](#), [IReadOnlyList<IntVector2>](#), [IReadOnlyCollection<IntVector2>](#), [IEnumerable<IntVector2>](#),  
[IEnumerable](#), [IEquatable<Line>](#)

## Inherited Members

[object.Equals\(object, object\)](#) ↗ , [object.GetType\(\)](#) ↗ , [object.MemberwiseClone\(\)](#) ↗ ,  
[object.ReferenceEquals\(object, object\)](#) ↗

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,

[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>,](#)  
[IEnumerable<T>\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>,](#)  
[Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>,](#)  
[Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>,](#)  
[params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,

[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#) ,  
[IReadOnlyListExtensions.GetRange<T>\(IReadOnlyList<T>, int, int\)](#) ,  
[IReadOnlyListExtensions.GetRange<T>\(IReadOnlyList<T>, Range\)](#) ,  
[IntRangeExtensions.GetRange<T>\(IReadOnlyList<T>, IntRange\)](#).

## Remarks

### Shape:

The [Line](#) will never contain 'right-angles':

```
#  
# #
```

The [Line](#) will always be 'perfect' if possible. By 'perfect', we mean it is drawn using blocks of a constant size. For example,

```
    # #  
    # #  
# #  
# #
```

The [Line](#) will always have 180-degree rotational symmetry, except potentially the midpoint in some odd-length [Lines](#).

Rotating the end point by a quadrant angle about the start point, or reflecting it across a vertical/horizontal axis through the start point, will apply the same transformation to the shape of the [Line](#). For example, with

```
# < end  
#  
# #  
#  
^start
```

rotating the end point 180 degrees about the start point gives the [Line](#)

```
# < start  
# #  
#  
#  
^end
```

Even though the x-distance and y-distance between the start and end haven't changed between the two [Lines](#), it matters which endpoint is the start and which is the end.

### **Enumerator:**

The enumerator starts at [start](#) and ends at [end](#).

The enumerator does not repeat any points.

## Constructors

### Line(IntVector2)

Creates a [Line](#) that's just a single point.

```
public Line(IntVector2 point)
```

#### Parameters

[point](#) [IntVector2](#)

### Line(IntVector2, IntVector2)

Creates a [Line](#) that starts at [start](#) and ends at [end](#).

```
public Line(IntVector2 start, IntVector2 end)
```

#### Parameters

[start](#) [IntVector2](#)

See [start](#).

end [IntVector2](#)

See [end](#).

## Properties

### Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

#### Property Value

[int](#)

The number of elements in the collection.

### this[int]

Returns the point on the [Line](#) at the given index, starting at [start](#).

```
public IntVector2 this[int index] { get; }
```

#### Parameters

[index](#) [int](#)

#### Property Value

[IntVector2](#)

#### Remarks

This is O(1).

#### Exceptions

[ArgumentOutOfRangeException](#)

`index` is < 0 or >= [Count](#).

## this[Range]

```
public IEnumerable<IntVector2> this[Range range] { get; }
```

Parameters

range [Range](#)

Property Value

[IEnumerable](#)<[IntVector2](#)>

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; }
```

Property Value

[IntRect](#)

## end

The end point of the [Line](#).

```
public IntVector2 end { get; set; }
```

Property Value

[IntVector2](#)

## See Also

[start](#)

## isHorizontal

True iff the [Line](#) has a constant y coord.

```
public bool isHorizontal { get; }
```

Property Value

[bool](#) ↗

### See Also

[isPoint](#), [isVertical](#)

## isPerfect

Whether this [Line](#) is divided into blocks of a constant size. E.g. a 12x4 [Line](#) is perfect as it is drawn as 4 horizontal blocks of 3 pixels.

```
public bool isPerfect { get; }
```

Property Value

[bool](#) ↗

## isPoint

True iff [start](#) and [end](#) are equal.

```
public bool isPoint { get; }
```

Property Value

[bool](#) ↗

### See Also

[isVertical](#), [isHorizontal](#)

## isVertical

True iff the [Line](#) has a constant x coord.

```
public bool isVertical { get; }
```

Property Value

[bool](#)

### See Also

[isPoint](#), [isHorizontal](#)

## reverse

Returns a new [Line](#) with [start](#) and [end](#) swapped.

This will look like the original [Line](#) rotated 180 degrees, which will look the same except potentially at the midpoint of some odd-length [Lines](#).

```
public Line reverse { get; }
```

Property Value

[Line](#)

## start

The start point of the [Line](#).

```
public IntVector2 start { get; set; }
```

Property Value

[IntVector2](#)

### See Also

[end](#)

# vector

end - start

```
public IntVector2 vector { get; }
```

Property Value

[IntVector2](#)

## Methods

### Contains(IntVector2)

```
public bool Contains(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#) ↗

Remarks

This is O(1).

### CountOnX(int)

Returns the number of pixels on the [Line](#) that have the given x coord.

```
public int CountOnX(int x)
```

Parameters

x [int](#) ↗

Returns

[int ↗](#)

Remarks

This is O(1).

**See Also**

[CountOnY\(int\)](#)

## CountOnY(int)

Returns the number of pixels on the [Line](#) that have the given y coord.

```
public int CountOnY(int y)
```

Parameters

y [int ↗](#)

Returns

[int ↗](#)

Remarks

This is O(1).

**See Also**

[CountOnX\(int\)](#)

## DeepCopy()

Returns a deep copy of the shape.

```
public Line DeepCopy()
```

Returns

## Equals(Line)

See [operator ==\(Line, Line\).](#)

```
public bool Equals(Line other)
```

Parameters

other [Line](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(Line\)](#)

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public Line Flip(FlipAxis axis)
```

Parameters

`axis` [FlipAxis](#)

Returns

[Line](#)

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumrator<IntVector2> GetEnumerator()
```

Returns

[IEnumrator](#)<[IntVector2](#)>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## MaxX(int)

Returns the maximum x coord of the pixels on the [Line](#) that have the given y coord.

```
public int MaxX(int y)
```

Parameters

*y* [int](#)

Returns

[int](#)

Remarks

This is O(1).

Exceptions

[ArgumentOutOfRangeException](#)

*y* is not in the range of y coords spanned by the [Line](#).

### See Also

[MinX\(int\)](#), [MaxY\(int\)](#), [MinY\(int\)](#)

## MaxY(int)

Returns the maximum y coord of the pixels on the [Line](#) that have the given x coord.

`public int MaxY(int x)`

Parameters

*x* [int](#)

Returns

[int](#)

Remarks

This is O(1).

Exceptions

## [ArgumentOutOfRangeException](#)

*x* is not in the range of x coords spanned by the [Line](#).

### See Also

[MinY\(int\)](#), [MaxX\(int\)](#), [MinX\(int\)](#).

## MinX(int)

Returns the minimum x coord of the pixels on the [Line](#) that have the given y coord.

```
public int MinX(int y)
```

### Parameters

*y* [int](#)

### Returns

[int](#)

### Remarks

This is O(1).

### Exceptions

## [ArgumentOutOfRangeException](#)

*y* is not in the range of y coords spanned by the [Line](#).

### See Also

[MaxX\(int\)](#), [MinY\(int\)](#), [MaxY\(int\)](#)

## MinY(int)

Returns the minimum y coord of the pixels on the [Line](#) that have the given x coord.

```
public int MinY(int x)
```

Parameters

x [int](#)

Returns

[int](#)

Remarks

This is O(1).

Exceptions

[ArgumentOutOfRangeException](#)

x is not in the range of x coords spanned by the [Line](#).

**See Also**

[MaxY\(int\)](#), [MinX\(int\)](#), [MaxX\(int\)](#)

## PointIsAbove(IntVector2)

Whether the point above / on the [Line](#) (and lies within the x range of the [Line](#)).

```
public bool PointIsAbove(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#)

Remarks

This is O(1).

**See Also**

[PointIsToLeft\(IntVector2\)](#), [PointIsToRight\(IntVector2\)](#), [PointIsBelow\(IntVector2\)](#)

## PointIsBelow(IntVector2)

Whether the point below / on the [Line](#) (and lies within the x range of the [Line](#)).

```
public bool PointIsBelow(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#) ↗

Remarks

This is O(1).

**See Also**

[PointIsToLeft\(IntVector2\)](#), [PointIsToRight\(IntVector2\)](#), [PointIsAbove\(IntVector2\)](#)

## PointIsToLeft(IntVector2)

Whether the point lies to the left of / on the [Line](#) (and lies within the y range of the [Line](#)).

```
public bool PointIsToLeft(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#) ↗

Remarks

This is O(1).

**See Also**

[PointIsToRight\(IntVector2\)](#), [PointIsBelow\(IntVector2\)](#), [PointIsAbove\(IntVector2\)](#)

## PointIsToRight(IntVector2)

Whether the point lies to the right of / on the [Line](#) (and lies within the y range of the [Line](#)).

```
public bool PointIsToRight(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#)

Remarks

This is O(1).

### See Also

[PointIsToLeft\(IntVector2\)](#), [PointIsBelow\(IntVector2\)](#), [PointIsAbove\(IntVector2\)](#)

## Rotate(RotationAngle)

Returns a deep copy of the shape rotated by the given angle.

```
public Line Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

[Line](#)

### See Also

operator [\\_\(IRotatableShape<T>\)](#)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public Line Translate(IntVector2 translation)
```

Parameters

[translation](#) [IntVector2](#)

Returns

[Line](#)

### See Also

operator [+\(ITranslatableShape<T>, IntVector2\)](#),

operator [+\(IntVector2, ITranslatableShape<T>\)](#),

operator [-\(ITranslatableShape<T>, IntVector2\)](#)

## Operators

### operator +(IntVector2, Line)

Returns a deep copy of the [Line](#) translated by the given vector.

```
public static Line operator +(IntVector2 translation, Line line)
```

Parameters

`translation` [IntVector2](#)

`line` [Line](#)

Returns

[Line](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator +(Line, IntVector2)

Returns a deep copy of the [Line](#) translated by the given vector.

```
public static Line operator +(Line line, IntVector2 translation)
```

Parameters

`line` [Line](#)

`translation` [IntVector2](#)

Returns

[Line](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator ==(Line, Line)

Whether the two [Lines](#) have the same [start](#) and [end](#).

```
public static bool operator ==(Line a, Line b)
```

Parameters

a [Line](#)

b [Line](#)

Returns

[bool](#) ↗

## operator !=(Line, Line)

See [operator ==\(Line, Line\).](#)

```
public static bool operator !=(Line a, Line b)
```

Parameters

a [Line](#)

b [Line](#)

Returns

[bool](#) ↗

## operator -(Line, IntVector2)

Returns a deep copy of the [Line](#) translated by the given vector.

```
public static Line operator -(Line line, IntVector2 translation)
```

Parameters

line [Line](#)

translation [IntVector2](#)

Returns

[Line](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator -(Line)

Returns a deep copy of the [Line](#) rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static Line operator -(Line line)
```

### Parameters

**line** [Line](#)

### Returns

[Line](#)

**See Also**

[Rotate\(RotationAngle\)](#), [Flip\(FlipAxis\)](#)

# Class Path

Namespace: [PAC.Shapes](#)

A sequence of connected [Lines](#). For example,

```
end of line 1
start of line 2      # < end of line 3
    v          #
    # #        #
    #      # #    # < start of line 3
    #          # #
    #                  ^
    ^          end of line 2
start of line 1
```

```
public class Path : I1DShape<Path>, ITransformableShape<Path>,
ITranslatableShape<Path>, IFlippableShape<Path>, IRotatableShape<Path>,
IDeepCopyableShape<Path>, IShape, IReadOnlyContains<IntVector2>,
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>,
IEnumerable, IEquatable<Path>
```

## Inheritance

[object](#) ↗ ← Path

## Implements

[I1DShape<Path>](#), [ITransformableShape<Path>](#), [ITranslatableShape<Path>](#),  
[IFlippableShape<Path>](#), [IRotatableShape<Path>](#), [IDeepCopyableShape<Path>](#), [IShape](#),  
[IReadOnlyContains<IntVector2>](#), [IReadOnlyCollection<IntVector2>](#),  
[IEnumerable<IntVector2>](#), [IEnumerable](#), [IEquatable<Path>](#)

## Inherited Members

[object.Equals\(object, object\)](#) ↗ , [object.GetType\(\)](#) ↗ , [object.MemberwiseClone\(\)](#) ↗ ,  
[object.ReferenceEquals\(object, object\)](#) ↗

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,

[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,

[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

The [Lines](#) must connect, meaning the end of each [Line](#) must be equal to or adjacent (including diagonally) to the start of the next [Line](#). This does not have to hold for the last [Line](#).

### Enumerator:

Points can be double-counted, but the purpose of [Path](#) is that they will never appear twice in succession. This allows iterating over the sequence of [Lines](#) without double-counting joining points between one [Line](#) and the next. Additionally, if the last point of the [Path](#) equals the first point, then the last point will not be counted, allowing iterating over a sequence of [Lines](#) that create a loop without double-counting the starting point.

## Constructors

### Path(params IntVector2[])

Creates a [Path](#) by connecting the sequence of points with [Lines](#).

```
public Path(params IntVector2[] points)
```

#### Parameters

`points` [IntVector2\[\]](#)

#### Exceptions

[ArgumentException](#)

`points` is empty.

## Path(params Line[])

Creates a [Path](#) with the given sequence of [Lines](#).

```
public Path(params Line[] lines)
```

### Parameters

**lines** [Line\[\]](#)

### Remarks

The [Lines](#) must connect, meaning the end of each [Line](#) must be equal to or adjacent to (including diagonally) the start of the next [Line](#).

Creates deep copies of the [Lines](#).

### Exceptions

[ArgumentException](#)

**lines** is empty or the [Lines](#) it contains do not connect.

## Path(IEnumerable<IntVector2>)

Creates a [Path](#) by connecting the sequence of points with [Lines](#).

```
public Path(IEnumerable<IntVector2> points)
```

### Parameters

**points** [IEnumerable](#)<[IntVector2](#)>

### Exceptions

[ArgumentException](#)

**points** is empty.

# Path(IEnumerable<Line>)

Creates a [Path](#) with the given sequence of [Lines](#).

```
public Path(IEnumerable<Line> lines)
```

## Parameters

`lines` [IEnumerable](#)<[Line](#)>

## Remarks

The [Lines](#) must connect, meaning the end of each [Line](#) must be equal to or adjacent to (including diagonally) the start of the next [Line](#).

Creates deep copies of the [Lines](#).

## Exceptions

[ArgumentException](#)

`lines` is empty or the [Lines](#) it contains do not connect.

# Properties

## Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

## Property Value

[int](#)

The number of elements in the collection.

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; }
```

Property Value

[IntRect](#)

end

The last point in the [Path](#).

```
public IntVector2 end { get; }
```

Property Value

[IntVector2](#)

Remarks

This may not be the last point in the enumerator: if it's equal to [start](#), and the enumerator has length > 1, it won't be counted at the end. Other than in that case, it will be the last point in the enumerator.

### See Also

[start](#)

## isHorizontal

True iff the [Path](#) never changes y coord.

```
public bool isHorizontal { get; }
```

Property Value

[bool](#)

### See Also

[isPoint](#), [isVertical](#)

## isLoop

Whether the [end](#) of the [Path](#) is adjacent (including diagonally) to the [start](#).

```
public bool isLoop { get; }
```

Property Value

[bool](#) ↗

## isPoint

True iff the [Path](#) never changes coordinate.

```
public bool isPoint { get; }
```

Property Value

[bool](#) ↗

### See Also

[isVertical](#), [isHorizontal](#)

## isVertical

True iff the [Path](#) never changes x coord.

```
public bool isVertical { get; }
```

Property Value

[bool](#) ↗

### See Also

[isPoint](#), [isHorizontal](#)

## lines

The sequence of [Lines](#) that make up the [Path](#).

```
public IReadOnlyCollection<Line> lines { get; }
```

Property Value

[ReadOnlyCollection](#)<[Line](#)>

Remarks

Do not edit these [Lines](#) except through methods provided in [Path](#), as editing them can potentially cause us to lose the property that the [Lines](#) connect.

## reverse

Returns a new [Path](#) going through the [Lines](#) in reverse order and with their starts / ends swapped.

```
public Path reverse { get; }
```

Property Value

[Path](#)

Remarks

This is not guaranteed to give the same geometry: the midpoint of a [Line](#) can move when swapping its start / end.

## selfIntersects

Whether the enumerator repeats any points.

```
public bool selfIntersects { get; }
```

Property Value

[bool](#)

## start

The first point in the [Path](#).

```
public IntVector2 start { get; }
```

### Property Value

[IntVector2](#)

### Remarks

This will be the first point in the enumerator.

### See Also

[end](#)

## Methods

### Concat(params Path[])

Puts the [Paths](#) one after the other to create a new [Path](#).

```
public static Path Concat(params Path[] paths)
```

### Parameters

paths [Path](#)[]

### Returns

[Path](#)

### Remarks

The [Paths](#) must connect, meaning the end of each [Path](#) must be equal to or adjacent to (including diagonally) the start of the next [Path](#).

Creates deep copies of the [Paths](#).

## Exceptions

### [ArgumentException](#)

`paths` is empty or the [Path](#)s it contains do not connect.

## Concat(IEnumerable<Path>)

Puts the [Paths](#) one after the other to create a new [Path](#).

```
public static Path Concat(IEnumerable<Path> paths)
```

## Parameters

`paths` [IEnumerable](#)<[Path](#)>

## Returns

[Path](#)

## Remarks

The [Paths](#) must connect, meaning the end of each [Path](#) must be equal to or adjacent to (including diagonally) the start of the next [Path](#).

Creates deep copies of the [Paths](#).

## Exceptions

### [ArgumentException](#)

`paths` is empty or the [Path](#)s it contains do not connect.

## Contains(IntVector2)

```
public bool Contains(IntVector2 point)
```

Parameters

point [IntVector2](#)

Returns

[bool](#)

## CountOnX(int)

Returns the number of points on the [Path](#) that have the given x coord.

```
public int CountOnX(int x)
```

Parameters

x [int](#)

Returns

[int](#)

Remarks

A point that appears n times in the enumerator will be counted n times.

## CountOnY(int)

Returns the number of points on the [Path](#) that have the given y coord.

```
public int CountOnY(int y)
```

Parameters

y [int](#)

Returns

[int](#)

## Remarks

A point that appears n times in the enumerator will be counted n times.

## DeepCopy()

Returns a deep copy of the shape.

```
public Path DeepCopy()
```

Returns

[Path](#)

## Equals(Path)

See [operator ==\(Path, Path\)](#).

```
public bool Equals(Path other)
```

Parameters

other [Path](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(Path\)](#).

```
public override bool Equals(object obj)
```

Parameters

[obj](#) [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public Path Flip(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

Returns

[Path](#)

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumarator<IntVector2> GetEnumerator()
```

Returns

[IEnumarator](#)<[IntVector2](#)>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## MaxX(int)

Returns the maximum x coord of the points on the [Path](#) that have the given y coord.

`public int MaxX(int y)`

Parameters

[y](#) [int](#)

Returns

[int](#)

Exceptions

[ArgumentOutOfRangeException](#)

[y](#) is not in the range of y coords spanned by the [Path](#).

## MaxY(int)

Returns the maximum y coord of the points on the [Path](#) that have the given x coord.

`public int MaxY(int x)`

Parameters

[x](#) [int](#)

Returns

[int](#)

## Exceptions

### [ArgumentOutOfRangeException](#)

`x` is not in the range of x coords spanned by the [Path](#).

## MinX(int)

Returns the minimum x coord of the points on the [Path](#) that have the given y coord.

```
public int MinX(int y)
```

## Parameters

`y` [int](#)

## Returns

[int](#)

## Exceptions

### [ArgumentOutOfRangeException](#)

`y` is not in the range of y coords spanned by the [Path](#).

## MinY(int)

Returns the minimum y coord of the points on the [Path](#) that have the given x coord.

```
public int MinY(int x)
```

## Parameters

`x` [int](#)

## Returns

[int](#)

## Exceptions

### [ArgumentOutOfRangeException](#)

x is not in the range of x coords spanned by the [Path](#).

## Rotate(RotationAngle)

Returns a deep copy of the shape rotated by the given angle.

```
public Path Rotate(RotationAngle angle)
```

### Parameters

angle [RotationAngle](#)

### Returns

[Path](#)

### See Also

operator -([IRotatableShape](#)<T>)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

### Returns

[string](#)

A string that represents the current object.

## Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public Path Translate(IntVector2 translation)
```

Parameters

translation [IntVector2](#)

Returns

[Path](#)

### See Also

operator [+\(ITranslatableShape<T>, IntVector2\)](#),  
operator [+\(IntVector2, ITranslatableShape<T>\)](#),  
operator [-\(ITranslatableShape<T>, IntVector2\)](#)

## Operators

### operator +(IntVector2, Path)

Returns a deep copy of the [Path](#) translated by the given vector.

```
public static Path operator +(IntVector2 translation, Path path)
```

Parameters

translation [IntVector2](#)

path [Path](#)

Returns

[Path](#)

### See Also

[Translate\(IntVector2\)](#)

## operator +(Path, IntVector2)

Returns a deep copy of the [Path](#) translated by the given vector.

```
public static Path operator +(Path path, IntVector2 translation)
```

Parameters

path [Path](#)

translation [IntVector2](#)

Returns

[Path](#)

### See Also

[Translate\(IntVector2\)](#)

## operator ==(Path, Path)

Whether the two [Paths](#) have the same sequence of [Lines](#) in [lines](#).

```
public static bool operator ==(Path a, Path b)
```

Parameters

a [Path](#)

b [Path](#)

Returns

[bool](#) ↗

## implicit operator Path(Line)

Treats the [Line](#) as a [Path](#) with one line segment.

```
public static implicit operator Path(Line line)
```

Parameters

line [Line](#)

Returns

[Path](#)

## operator !=(Path, Path)

See [operator ==\(Path, Path\)](#).

```
public static bool operator !=(Path a, Path b)
```

Parameters

a [Path](#)

b [Path](#)

Returns

[bool](#) ↴

## operator -(Path, IntVector2)

Returns a deep copy of the [Path](#) translated by the given vector.

```
public static Path operator -(Path path, IntVector2 translation)
```

Parameters

path [Path](#)

translation [IntVector2](#)

Returns

[Path](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator -(Path)

Returns a deep copy of the [Path](#) rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static Path operator -(Path path)
```

Parameters

**path** [Path](#)

Returns

[Path](#)

**See Also**

[Rotate\(RotationAngle\)](#), [Flip\(FlipAxis\)](#)

# Class Rectangle

Namespace: [PAC.Shapes](#)

A pixel art rectangle shape. For example,

```
# # # #
#      #
# # # #
```

```
public class Rectangle : I2DShape<Rectangle>, IFillableShape,
ITransformableShape<Rectangle>, ITranslatableShape<Rectangle>,
IFlippableShape<Rectangle>, IRotatableShape<Rectangle>,
IDeepCopyableShape<Rectangle>, IShape, IReadOnlyContains<IntVector2>,
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>,
IEnumerable, IEquatable<Rectangle>
```

## Inheritance

[object](#) ← Rectangle

## Implements

[I2DShape<Rectangle>](#), [IFillableShape](#), [ITransformableShape<Rectangle>](#),  
[ITranslatableShape<Rectangle>](#), [IFlippableShape<Rectangle>](#),  
[IRotatableShape<Rectangle>](#), [IDeepCopyableShape<Rectangle>](#), [IShape](#),  
[IReadOnlyContains<IntVector2>](#), [IReadOnlyCollection<IntVector2>](#),  
[IEnumerable<IntVector2>](#), [IEnumerable](#), [IEquatable<Rectangle>](#)

## Inherited Members

[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,

[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,

[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Remarks

### Enumerator:

The enumerator does not repeat any points.

## Constructors

### Rectangle(IntRect, bool)

Creates a [Rectangle](#) that takes up the region of the given [IntRect](#).

```
public Rectangle(IntRect boundingRect, bool filled)
```

#### Parameters

boundingRect [IntRect](#)

See [boundingRect](#).

filled [bool](#) ↗

See [filled](#).

## Properties

### Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

## Property Value

## [int](#)

The number of elements in the collection.

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; set; }
```

Property Value

[IntRect](#)

## filled

Whether the shape has its inside filled-in, or whether it's just the border.

```
public bool filled { get; set; }
```

Property Value

[bool](#)

## isSquare

Whether the [Rectangle](#) is a square.

```
public bool isSquare { get; }
```

Property Value

[bool](#)

## Remarks

This is equivalent to its [boundingRect](#) being a square.

# Methods

## Contains(IntVector2)

```
public bool Contains(IntVector2 point)
```

### Parameters

point [IntVector2](#)

### Returns

[bool](#)

## DeepCopy()

Returns a deep copy of the shape.

```
public Rectangle DeepCopy()
```

### Returns

[Rectangle](#)

## Equals(Rectangle)

See [operator ==\(Rectangle, Rectangle\)](#).

```
public bool Equals(Rectangle other)
```

### Parameters

other [Rectangle](#)

### Returns

[bool](#)

## Equals(object)

See [Equals\(Rectangle\)](#).

```
public override bool Equals(object obj)
```

Parameters

obj [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public Rectangle Flip(FlipAxis axis)
```

Parameters

axis [FlipAxis](#)

Returns

[Rectangle](#)

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<IntVector2> GetEnumerator()
```

Returns

## [IEnumerator](#) <IntVector2>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## Rotate(RotationAngle)

Returns a deep copy of the shape rotated by the given angle.

```
public Rectangle Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

[Rectangle](#)

### See Also

operator -([IRotatableShape](#)<T>)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public Rectangle Translate(IntVector2 translation)
```

Parameters

translation [IntVector2](#)

Returns

[Rectangle](#)

### See Also

operator [+\(ITranslatableShape<T>, IntVector2\)](#),  
operator [+\(IntVector2, ITranslatableShape<T>\)](#),  
operator [-\(ITranslatableShape<T>, IntVector2\)](#)

## Operators

### operator +(IntVector2, Rectangle)

Returns a deep copy of the [Rectangle](#) translated by the given vector.

```
public static Rectangle operator +(IntVector2 translation, Rectangle rectangle)
```

Parameters

`translation` [IntVector2](#)

`rectangle` [Rectangle](#)

Returns

[Rectangle](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator +(Rectangle, IntVector2)

Returns a deep copy of the [Rectangle](#) translated by the given vector.

```
public static Rectangle operator +(Rectangle rectangle, IntVector2 translation)
```

Parameters

`rectangle` [Rectangle](#)

`translation` [IntVector2](#)

Returns

[Rectangle](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator ==(Rectangle, Rectangle)

Whether the two [Rectangles](#) have the same shape, and the same value for [filled](#).

```
public static bool operator ==(Rectangle a, Rectangle b)
```

Parameters

`a` [Rectangle](#)

b [Rectangle](#)

Returns

[bool](#)

## operator !=(Rectangle, Rectangle)

See [operator ==\(Rectangle, Rectangle\)](#).

```
public static bool operator !=(Rectangle a, Rectangle b)
```

Parameters

a [Rectangle](#)

b [Rectangle](#)

Returns

[bool](#)

## operator -(Rectangle, IntVector2)

Returns a deep copy of the [Rectangle](#) translated by the given vector.

```
public static Rectangle operator -(Rectangle rectangle, IntVector2 translation)
```

Parameters

rectangle [Rectangle](#)

translation [IntVector2](#)

Returns

[Rectangle](#)

## See Also

[Translate\(IntVector2\)](#)

## operator -(Rectangle)

Returns a deep copy of the [Rectangle](#) rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static Rectangle operator -(Rectangle rectangle)
```

### Parameters

rectangle [Rectangle](#)

### Returns

[Rectangle](#)

### See Also

[Rotate\(RotationAngle\)](#), [Flip\(FlipAxis\)](#)

# Class RightTriangle

Namespace: [PAC.Shapes](#)

A pixel art right-angled triangle shape. For example,

```
# #
# # #
#
# #
# #
# # # # # # #
```

```
public class RightTriangle : I2DShape<RightTriangle>, IFillableShape,
ITransformableShape<RightTriangle>, ITranslatableShape<RightTriangle>,
IFlippableShape<RightTriangle>, IRotatableShape<RightTriangle>,
IDeepCopyableShape<RightTriangle>, IShape, IReadOnlyContains<IntVector2>,
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>,
IEnumerable, IEquatable<RightTriangle>
```

## Inheritance

[object](#) ← RightTriangle

## Implements

[I2DShape<RightTriangle>](#), [IFillableShape](#), [ITransformableShape<RightTriangle>](#),  
[ITranslatableShape<RightTriangle>](#), [IFlippableShape<RightTriangle>](#),  
[IRotatableShape<RightTriangle>](#), [IDeepCopyableShape<RightTriangle>](#), [IShape](#),  
[IReadOnlyContains<IntVector2>](#), [IReadOnlyCollection<IntVector2>](#),  
[IEnumerable<IntVector2>](#), [IEnumerable](#), [IEquatable<RightTriangle>](#)

## Inherited Members

[object.Equals\(object, object\)](#), [object.GetType\(\)](#), [object.MemberwiseClone\(\)](#),  
[object.ReferenceEquals\(object, object\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,

[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,

[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

### Shape:

The hypotenuse is drawn as a [Line](#). We draw this [Line](#) starting from the longer side of the [RightTriangle](#) (or the horizontal side if both sides have the same length). This is done to ensure that rotating/reflecting the corners doesn't change the geometry (up to rotating/reflecting).

Except in some special cases, this [Line](#) is not drawn from the corners. Instead, the endpoints are inset by 1 from the corners. For example,

```
line end
      v
      # #
# #   #
#       #
# #     #
line start > #
      #
# # # # # # #
```

This is done to make it look more aesthetic.

The special cases where the endpoints are not inset are:

- Width or height 1 - the [RightTriangle](#) looks like a straight line
- Width or height 2 - the [RightTriangle](#) looks like, for example,

```
# 
#
# #
# #
# #
```

where, if the width is 2, the height of the shorter vertical block is `ceil(height of triangle / 2)`, and analogously for when the height is 2.

## Enumerator:

The enumerator does not repeat any points.

# Constructors

## RightTriangle(IntRect, RightAngleLocation, bool)

Creates the largest [RightTriangle](#) that can fit in the given rect, and that has the right angle in the specified corner of the rect.

```
public RightTriangle(IntRect boundingRect, RightTriangle.RightAngleLocation  
rightAngleLocation, bool filled)
```

### Parameters

`boundingRect` [IntRect](#)

See [boundingRect](#).

`rightAngleLocation` [RightTriangle.RightAngleLocation](#)

See [rightAngleLocation](#).

`filled` [bool](#) ↗

See [filled](#).

## RightTriangle(IntVector2, IntVector2, bool, bool)

Creates a [RightTriangle](#) with the two given non-right-angle corners.

```
public RightTriangle(IntVector2 corner, IntVector2 oppositeCorner, bool  
horizontalEdgeIsBottomEdge, bool filled)
```

### Parameters

`corner` [IntVector2](#)

One of the two non-right-angle corners of the [RightTriangle](#).

`oppositeCorner` [IntVector2](#)

The non-right-angle corner opposite `corner`.

`horizontalEdgeIsBottomEdge` [bool](#)

Whether the horizontal edge of the [RightTriangle](#) is the bottom edge (as opposed to the top edge).

`filled` [bool](#)

See [filled](#).

## Properties

### Count

Gets the number of elements in the collection.

```
public int Count { get; }
```

### Property Value

[int](#)

The number of elements in the collection.

### bottomCorner

The lower of the two corners that don't contain the right angle.

```
public IntVector2 bottomCorner { get; }
```

### Property Value

[IntVector2](#)

## Remarks

This will be different from [topCorner](#), unless the [RightTriangle](#) is a single point.

## See Also

[topCorner](#), [leftCorner](#), [rightCorner](#), [rightAngleCorner](#)

## boundingRect

The smallest rect containing the whole shape.

```
public IntRect boundingRect { get; set; }
```

## Property Value

[IntRect](#)

## filled

Whether the shape has its inside filled-in, or whether it's just the border.

```
public bool filled { get; set; }
```

## Property Value

[bool](#)

## isIsosceles

Whether the [RightTriangle](#) is an isosceles triangle.

```
public bool isIsosceles { get; }
```

## Property Value

[bool](#)

## Remarks

This is equivalent to its [boundingRect](#) being a square.

## leftCorner

The left-most of the two corners that don't contain the right angle.

```
public IntVector2 leftCorner { get; }
```

### Property Value

[IntVector2](#)

## Remarks

This will be different from [rightCorner](#), unless the [RightTriangle](#) is a single point.

## See Also

[bottomCorner](#), [topCorner](#), [rightCorner](#), [rightAngleCorner](#)

## rightAngleCorner

The corner that contains the right angle.

```
public IntVector2 rightAngleCorner { get; }
```

### Property Value

[IntVector2](#)

## See Also

[bottomCorner](#), [topCorner](#), [leftCorner](#), [rightCorner](#)

## rightAngleLocation

Which corner of the [boundingRect](#) the right angle is in.

```
public RightTriangle.RightAngleLocation rightAngleLocation { get; set; }
```

Property Value

[RightTriangle.RightAngleLocation](#)

## rightCorner

The right-most of the two corners that don't contain the right angle.

```
public IntVector2 rightCorner { get; }
```

Property Value

[IntVector2](#)

Remarks

This will be different from [leftCorner](#), unless the [RightTriangle](#) is a single point.

**See Also**

[bottomCorner](#), [topCorner](#), [leftCorner](#), [rightAngleCorner](#)

## topCorner

The higher of the two corners that don't contain the right angle.

```
public IntVector2 topCorner { get; }
```

Property Value

[IntVector2](#)

Remarks

This will be different from [bottomCorner](#), unless the [RightTriangle](#) is a single point.

**See Also**

[bottomCorner](#), [leftCorner](#), [rightCorner](#), [rightAngleCorner](#)

## Methods

### Contains(IntVector2)

```
public bool Contains(IntVector2 point)
```

#### Parameters

point [IntVector2](#)

#### Returns

[bool](#)

### DeepCopy()

Returns a deep copy of the shape.

```
public RightTriangle DeepCopy()
```

#### Returns

[RightTriangle](#)

### Equals(RightTriangle)

See [operator ==\(RightTriangle, RightTriangle\)](#).

```
public bool Equals(RightTriangle other)
```

#### Parameters

other [RightTriangle](#)

Returns

[bool](#)

## Equals(object)

See [Equals\(RightTriangle\)](#).

```
public override bool Equals(object obj)
```

Parameters

[obj](#) [object](#)

Returns

[bool](#)

## Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

```
public RightTriangle Flip(FlipAxis axis)
```

Parameters

[axis](#) [FlipAxis](#)

Returns

[RightTriangle](#)

## GetEnumerator()

Returns an enumerator that iterates through the collection.

```
public IEnumerator<IntVector2> GetEnumerator()
```

Returns

[IEnumerator](#)<IntVector2>

An enumerator that can be used to iterate through the collection.

## GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

[int](#)

A hash code for the current object.

## Rotate(RotationAngle)

Returns a deep copy of the shape rotated by the given angle.

```
public RightTriangle Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

[RightTriangle](#)

### See Also

operator -([IRotatableShape](#)<T>)

## ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current object.

## Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public RightTriangle Translate(IntVector2 translation)
```

Parameters

[translation](#) [IntVector2](#)

Returns

[RightTriangle](#)

### See Also

operator [+\(ITranslatableShape<T>, IntVector2\)](#),

operator [+\(IntVector2, ITranslatableShape<T>\)](#),

operator [-\(ITranslatableShape<T>, IntVector2\)](#)

## Operators

### operator +(IntVector2, RightTriangle)

Returns a deep copy of the [RightTriangle](#) translated by the given vector.

```
public static RightTriangle operator +(IntVector2 translation,  
RightTriangle triangle)
```

## Parameters

translation [IntVector2](#)

triangle [RightTriangle](#)

## Returns

[RightTriangle](#)

### See Also

[Translate\(IntVector2\)](#)

## operator +(RightTriangle, IntVector2)

Returns a deep copy of the [RightTriangle](#) translated by the given vector.

```
public static RightTriangle operator +(RightTriangle triangle,  
IntVector2 translation)
```

## Parameters

triangle [RightTriangle](#)

translation [IntVector2](#)

## Returns

[RightTriangle](#)

### See Also

[Translate\(IntVector2\)](#)

## operator ==(RightTriangle, RightTriangle)

Whether the two [RightTriangles](#) have the same shape, and the same [rightAngleLocation](#).

```
public static bool operator ==(RightTriangle a, RightTriangle b)
```

Parameters

a [RightTriangle](#)

b [RightTriangle](#)

Returns

[bool](#)

## operator !=(RightTriangle, RightTriangle)

See [operator ==\(RightTriangle, RightTriangle\).](#)

```
public static bool operator !=(RightTriangle a, RightTriangle b)
```

Parameters

a [RightTriangle](#)

b [RightTriangle](#)

Returns

[bool](#)

## operator -(RightTriangle, IntVector2)

Returns a deep copy of the [RightTriangle](#) translated by the given vector.

```
public static RightTriangle operator -(RightTriangle triangle,  
IntVector2 translation)
```

Parameters

triangle [RightTriangle](#)

translation [IntVector2](#)

Returns

[RightTriangle](#)

**See Also**

[Translate\(IntVector2\)](#)

## operator -(RightTriangle)

Returns a deep copy of the [RightTriangle](#) rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static RightTriangle operator -(RightTriangle triangle)
```

Parameters

triangle [RightTriangle](#)

Returns

[RightTriangle](#)

**See Also**

[Rotate\(RotationAngle\)](#), [Flip\(FlipAxis\)](#)

# Enum RightTriangle.RightAngleLocation

Namespace: [PAC.Shapes](#)

Which corner of the [boundingRect](#) the right angle is in.

```
public enum RightTriangle.RightAngleLocation
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

BottomLeft = 0

BottomRight = 1

TopLeft = 2

TopRight = 3

# Namespace PAC.Shapes.Interfaces

## Interfaces

### [I1DShape<T>](#)

A pixel art representation of a 1D shape (such as a straight line, or a curve).

### [I2DShape<T>](#)

A pixel art representation of a 2D shape (such as a rectangle or a circle).

### [IDeepCopyableShape<T>](#)

An [IShape](#) that can be deep-copied.

### [IFillableShape](#)

An [IShape](#) that encloses a region that can be filled in.

### [IFlippableShape<T>](#)

An [IShape](#) that can be flipped.

### [IIsoMetricShape<T>](#)

A pixel art representation of the isometric projection of a shape.

### [IRotatableShape<T>](#)

An [IShape](#) that can be rotated.

### [IShape](#)

Defines a non-empty collection of integer points that make up a pixel art shape. The shape must be connected (including diagonally).

### [ITransformableShape<T>](#)

An [IShape](#) that can be translated, flipped and rotated.

### [ITranslatableShape<T>](#)

An [IShape](#) that can be translated.

# Interface I1DShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

A pixel art representation of a 1D shape (such as a straight line, or a curve).

```
public interface I1DShape<out T> : ITransformableShape<T>, ITranslatableShape<T>,  
IFlippableShape<T>, IRotatableShape<T>, IDeepCopyableShape<T>, IShape,  
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,  
IEnumerable<IntVector2>, IEnumerable where T : IShape
```

## Type Parameters

T

The type of shape obtained from transformations on the shape. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[ITranslatableShape<T>.Translate\(IntVector2\)](#) , [IFlippableShape<T>.Flip\(FlipAxis\)](#) ,  
[IRotatableShape<T>.Rotate\(RotationAngle\)](#) , [IDeepCopyableShape<T>.DeepCopy\(\)](#) ,  
[IShape.boundingRect](#) , [IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,

[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, I\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Remarks

See [ITranslatableShape<T>](#) for details of the design pattern regarding the generic type parameter.

# Interface I2DShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

A pixel art representation of a 2D shape (such as a rectangle or a circle).

```
public interface I2DShape<out T> : IFillableShape, ITransformableShape<T>,  
ITranslatableShape<T>, IFlippableShape<T>, IRotatableShape<T>,  
IDeepCopyableShape<T>, IShape, IReadOnlyContains<IntVector2>,  
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>, IEnumerable where T  
: IShape
```

## Type Parameters

T

The type of shape obtained from transformations on the shape. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[IFillableShape.filled](#) , [ITranslatableShape<T>.Translate\(IntVector2\)](#) ,  
[IFlippableShape<T>.Flip\(FlipAxis\)](#) , [IRotatableShape<T>.Rotate\(RotationAngle\)](#) ,  
[IDeepCopyableShape<T>.DeepCopy\(\)](#) , [IShape.boundingRect](#) ,  
[IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) ↴ , [IEnumerable<IntVector2>.GetEnumerator\(\)](#) ↴

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,

[IShapeExtensions.ToTexture\(I Shape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(I Shape, Color, IntRect\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,

[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Remarks

See [ITranslatableShape<T>](#) for details of the design pattern regarding the generic type parameter.

# Interface IDeepCopyableShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

An [IShape](#) that can be deep-copied.

```
public interface IDeepCopyableShape<out T> : IShape, IReadOnlyContains<IntVector2>,  
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>, IEnumerable where T  
: IShape
```

## Type Parameters

T

The type of shape obtained from the deep copy. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[IShape.boundingRect](#) , [IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>,  
IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>,  
IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>,  
Func<TElement, TCompare>\)](#) ,

[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#),  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#),  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#),  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#),  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

See [ITranslatableShape<T>](#) for details of the design pattern regarding the generic type parameter.

# Methods

## DeepCopy()

Returns a deep copy of the shape.

T **DeepCopy()**

Returns

T

# Interface IFillableShape

Namespace: [PAC.Shapes.Interfaces](#)

An [IShape](#) that encloses a region that can be filled in.

```
public interface IFillableShape : IShape, IReadOnlyContains<IntVector2>,  
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>, IEnumerable
```

## Inherited Members

[IShape.boundingRect](#) , [IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

```
IReadOnlyContainsExtensions.ContainsAll<T>(IReadOnlyContains<T>, params T[]) ,  
IReadOnlyContainsExtensions.ContainsAll<T>(IReadOnlyContains<T>, IEnumerable<T>) ,  
IReadOnlyContainsExtensions.ContainsAny<T>(IReadOnlyContains<T>, params T[]) ,  
IReadOnlyContainsExtensions.ContainsAny<T>(IReadOnlyContains<T>, IEnumerable<T>) ,  
IReadOnlyContainsExtensions.ContainsNone<T>(IReadOnlyContains<T>, params T[]) ,  
IReadOnlyContainsExtensions.ContainsNone<T>(IReadOnlyContains<T>,  
IEnumerable<T>) ,  
IReadOnlyContainsExtensions.ContainsNotAll<T>(IReadOnlyContains<T>, params T[]) ,  
IReadOnlyContainsExtensions.ContainsNotAll<T>(IReadOnlyContains<T>,  
IEnumerable<T>) ,  
ISetComparableExtensions.IsProperSubsetOf<T1, T2>(T1, T2) ,  
ISetComparableExtensions.IsProperSupersetOf<T1, T2>(T1, T2) ,  
IShapeExtensions.ToTexture(IShape, Color) ,  
IShapeExtensions.ToTexture(IShape, Color, IntRect) ,  
IEnumerableExtensions.AreAllDistinct<T>(IEnumerable<T>) ,  
IEnumerableExtensions.ArgMax<TElement, TCompare>(IEnumerable<TElement>,  
Func<TElement, TCompare>) ,  
IEnumerableExtensions.ArgMin<TElement, TCompare>(IEnumerable<TElement>,  
Func<TElement, TCompare>) ,  
IEnumerableExtensions.Concat<T>(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>,  
params IEnumerable<T>[]) ,  
IEnumerableExtensions.ContainsAll<T>(IEnumerable<T>, params T[]) ,  
IEnumerableExtensions.ContainsAll<T>(IEnumerable<T>, IEnumerable<T>) ,  
IEnumerableExtensions.ContainsAny<T>(IEnumerable<T>, params T[]) ,  
IEnumerableExtensions.ContainsAny<T>(IEnumerable<T>, IEnumerable<T>) ,  
IEnumerableExtensions.ContainsNone<T>(IEnumerable<T>, params T[]) ,
```

[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Properties

### filled

Whether the shape has its inside filled-in, or whether it's just the border.

```
bool filled { get; set; }
```

### Property Value

[bool](#) ↗



# Interface IFlippableShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

An [IShape](#) that can be flipped.

```
public interface IFlippableShape<out T> : IDeepCopyableShape<T>, IShape,  
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,  
IEnumerable<IntVector2>, IEnumerable where T : IShape
```

## Type Parameters

T

The type of shape obtained from flipping. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[IDeepCopyableShape<T>.DeepCopy\(\)](#) , [IShape.boundingRect](#) ,  
[IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,

[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#),  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#),  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#),  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#),  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

See [ITranslatableShape<T>](#) for details of the design pattern regarding the generic type parameter.

## Methods

### Flip(FlipAxis)

Returns a deep copy of the shape reflected across the given axis.

T **Flip**(FlipAxis axis)

#### Parameters

axis [FlipAxis](#)

#### Returns

T

# Interface IIsometricShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

A pixel art representation of the isometric projection of a shape.

```
public interface IIsoMetricShape<out T> : IFillableShape, ITranslatableShape<T>,  
IFlippableShape<T>, IDepcopyableShape<T>, IShape, IReadOnlyContains<IntVector2>,  
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>, IEnumerable where T  
: IShape
```

## Type Parameters

T

The type of shape obtained from transformations on the shape. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[IFillableShape.filled](#) , [ITranslatableShape<T>.Translate\(IntVector2\)](#) ,  
[IFlippableShape<T>.Flip\(FlipAxis\)](#) , [IDeepCopyableShape<T>.DeepCopy\(\)](#) ,  
[IShape.boundingRect](#) , [IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,

[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, I\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Remarks

See [ITranslatableShape<T>](#) for details of the design pattern regarding the generic type parameter.

# Interface IRotatableShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

An [IShape](#) that can be rotated.

```
public interface IRotatableShape<out T> : IDeepCopyableShape<T>, IShape,  
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,  
IEnumerable<IntVector2>, IEnumerable where T : IShape
```

## Type Parameters

T

The type of shape obtained from rotating. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[IDeepCopyableShape<T>.DeepCopy\(\)](#) , [IShape.boundingRect](#) ,  
[IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,

[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#),  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#),  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#),  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#),  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

See [ITranslatableShape<T>](#) for details of the design pattern regarding the generic type parameter.

## Methods

### Rotate(RotationAngle)

Returns a deep copy of the shape rotated by the given angle.

```
T Rotate(RotationAngle angle)
```

Parameters

angle [RotationAngle](#)

Returns

T

### See Also

[operator -\(IRotatableShape<T>\)](#)

## Operators

### operator -(IRotatableShape<T>)

Returns a deep copy of the shape rotated 180 degrees about the origin (equivalently, reflected through the origin).

```
public static T operator -(IRotatableShape<out T> shape)
```

Parameters

shape [IRotatableShape<T>](#)

Returns

T

**See Also**

[Rotate\(RotationAngle\)](#)

# Interface IShape

Namespace: [PAC.Shapes.Interfaces](#)

Defines a non-empty collection of integer points that make up a pixel art shape. The shape must be connected (including diagonally).

```
public interface IShape : IReadOnlyContains<IntVector2>,
IReadOnlyCollection<IntVector2>, IEnumerable<IntVector2>, IEnumerable
```

## Inherited Members

[IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,

[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Remarks

Though uncommon among the shapes that implement this interface, points *can* be repeated in the collection. Whether this can happen or not should be documented for each implementing shape. Note that [Count](#) should include multiplicities.

## Properties

### boundingRect

The smallest rect containing the whole shape.

```
IntRect boundingRect { get; }
```

Property Value

[IntRect](#)

# Interface ITransformableShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

An [IShape](#) that can be translated, flipped and rotated.

```
public interface ITransformableShape<out T> : ITranslatableShape<T>,  
IFlippableShape<T>, IRotatableShape<T>, IDeepCopyableShape<T>, IShape,  
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,  
IEnumerable<IntVector2>, IEnumerable where T : IShape
```

## Type Parameters

T

The type of shape obtained from the transformations. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[ITranslatableShape<T>.Translate\(IntVector2\)](#) , [IFlippableShape<T>.Flip\(FlipAxis\)](#) ,  
[IRotatableShape<T>.Rotate\(RotationAngle\)](#) , [IDeepCopyableShape<T>.DeepCopy\(\)](#) ,  
[IShape.boundingRect](#) , [IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,

[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#) ,  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#) ,  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#) ,  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, I\)](#) ,  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#) ,  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#) ,  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#) ,  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#) ,  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#) ,  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#) ,  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#)

## Remarks

See [ITranslatableShape<T>](#) for details of the design pattern regarding the generic type parameter.

# Interface ITranslatableShape<T>

Namespace: [PAC.Shapes.Interfaces](#)

An [IShape](#) that can be translated.

```
public interface ITranslatableShape<out T> : IDeepCopyableShape<T>, IShape,  
IReadOnlyContains<IntVector2>, IReadOnlyCollection<IntVector2>,  
IEnumerable<IntVector2>, IEnumerable where T : IShape
```

## Type Parameters

T

The type of shape obtained from translating. When implementing this interface on a concrete type, this should be the same as the implementing type. See [ITranslatableShape<T>](#) for more detail on this design pattern.

## Inherited Members

[IDeepCopyableShape<T>.DeepCopy\(\)](#) , [IShape.boundingRect](#) ,  
[IReadOnlyContains<IntVector2>.Contains\(IntVector2\)](#) ,  
[IReadOnlyCollection<IntVector2>.Count](#) , [IEnumerable<IntVector2>.GetEnumerator\(\)](#)

## Extension Methods

[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsAny<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNone<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, params T\[\]\)](#) ,  
[IReadOnlyContainsExtensions.ContainsNotAll<T>\(IReadOnlyContains<T>, IEnumerable<T>\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color\)](#) ,  
[IShapeExtensions.ToTexture\(IShape, Color, IntRect\)](#) ,  
[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#) ,  
[IEnumerableExtensions.AreAllDistinct<T>\(IEnumerable<T>\)](#) ,

[IEnumerableExtensions.ArgMax<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.ArgMin<TElement, TCompare>\(IEnumerable<TElement>, Func<TElement, TCompare>\)](#),  
[IEnumerableExtensions.Concat<T>\(IEnumerable<T>, IEnumerable<T>, IEnumerable<T>, params IEnumerable<T>\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsAny<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNone<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, params T\[\]\)](#),  
[IEnumerableExtensions.ContainsNotAll<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.CountIsAtLeast<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsAtMost<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.CountIsExactly<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.EnumerateOccurrences<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Enumerate<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSubsequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.IsSupersequenceOf<T>\(IEnumerable<T>, IEnumerable<T>\)](#),  
[IEnumerableExtensions.MinAndMax<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.None<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.NotAll<T>\(IEnumerable<T>, Func<T, bool>\)](#),  
[IEnumerableExtensions.Replace<T>\(IEnumerable<T>, T, T\)](#),  
[IEnumerableExtensions.SkipIndex<T>\(IEnumerable<T>, int\)](#),  
[IEnumerableExtensions.ToPrettyString<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.ZipCurrentAndNext<T>\(IEnumerable<T>\)](#),  
[IEnumerableExtensions.Zip<T1, T2>\(IEnumerable<T1>, IEnumerable<T2>\)](#),  
[Combinatorics.Pairs<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Pairs<TFirst, TSecond>\(IEnumerable<TFirst>, IEnumerable<TSecond>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Permutations<T>\(IEnumerable<T>, int\)](#),  
[Combinatorics.Triples<T>\(IEnumerable<T>\)](#),  
[Combinatorics.Triples<TFirst, TSecond, TThird>\(IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>\)](#),  
[Combinatorics.Tuples<T>\(IEnumerable<T>, int\)](#).

## Remarks

When implementing this interface on a concrete type, the type parameter `T` should be the same as the implementing type. For example,

```
class Point : ITranslatableShape<Point>
{
    public IntVector2 point;

    public Point(IntVector2 point)
    {
        this.point = point;
    }

    public Point Translate(IntVector2 translation) => new Point(point
+ translation);

    // Implementation of the rest of the interface...
}
```

This is done to ensure the specific return type of the `Translate(IntVector2)` method. Note though that `T` is covariant (see [out ↴](#)), so you can abstract over any shape that is translatable. For example,

```
List<ITranslatableShape<IShape>>
```

This can store any shape that implements `ITranslatableShape<T>`. Note though that calling `Translate(IntVector2)` on an element of this list will return an `IShape` so cannot be translated again without downcasting. However you can do, say,

```
List<ITranslatableShape<ITranslatableShape<IShape>>>
```

This allows you to translate twice before you get an `IShape` returned.

## Methods

### Translate(IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
T Translate(IntVector2 translation)
```

Parameters

translation [IntVector2](#)

Returns

T

**See Also**

[operator +\(ITranslatableShape<T>, IntVector2\)](#),

[operator +\(IntVector2, ITranslatableShape<T>\)](#),

[operator -\(ITranslatableShape<T>, IntVector2\)](#)

## Operators

### operator +(IntVector2, ITranslatableShape<T>)

Returns a deep copy of the shape translated by the given vector.

```
public static T operator +(IntVector2 translation, ITranslatableShape<out T> shape)
```

Parameters

translation [IntVector2](#)

shape [ITranslatableShape<T>](#)

Returns

T

**See Also**

[Translate\(IntVector2\)](#)

### operator +(ITranslatableShape<T>, IntVector2)

Returns a deep copy of the shape translated by the given vector.

```
public static T operator +(ITranslatableShape<out T> shape, IntVector2 translation)
```

Parameters

shape [ITranslatableShape<T>](#)

translation [IntVector2](#)

Returns

T

**See Also**

[Translate\(IntVector2\)](#)

**operator -(ITranslatableShape<T>, IntVector2)**

Returns a deep copy of the shape translated by the given vector.

```
public static T operator -(ITranslatableShape<out T> shape, IntVector2 translation)
```

Parameters

shape [ITranslatableShape<T>](#)

translation [IntVector2](#)

Returns

T

**See Also**

[Translate\(IntVector2\)](#)

# Namespace PAC.Themes

## Classes

[Theme](#)

[ThemeManager](#)

[UITheme](#)

## Enums

[ThemeObjectType](#)

# Class Theme

Namespace: [PAC.Themes](#)

```
public class Theme : ScriptableObject
```

## Inheritance

[object](#) ← Theme

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### backgroundColour

```
public Color backgroundColour
```

Field Value

Color

### buttonColour

```
public Color buttonColour
```

Field Value

Color

### buttonHoverColour

```
public Color buttonHoverColour
```

Field Value

Color

## buttonPressedColour

```
public Color buttonPressedColour
```

Field Value

Color

## layerTileHoverTint

```
public Color layerTileHoverTint
```

Field Value

Color

## layerTileOffColour

```
public Color layerTileOffColour
```

Field Value

Color

## layerTileOnColour

```
public Color layerTileOnColour
```

Field Value

Color

## layerTilePressedColour

```
public Color layerTilePressedColour
```

Field Value

Color

## panelColour

```
public Color panelColour
```

Field Value

Color

## scaleColour

```
public Color scaleColour
```

Field Value

Color

## scaleHoverColour

```
public Color scaleHoverColour
```

Field Value

Color

## scalePressedColour

```
public Color scalePressedColour
```

Field Value

Color

## scrollbarBackgroundColour

```
public Color scrollbarBackgroundColour
```

Field Value

Color

## scrollbarBackgroundHoverColour

```
public Color scrollbarBackgroundHoverColour
```

Field Value

Color

## scrollbarBackgroundPressedColour

```
public Color scrollbarBackgroundPressedColour
```

Field Value

Color

## scrollbarHandleColour

```
public Color scrollbarHandleColour
```

Field Value

Color

## scrollbarHandleHoverColour

```
public Color scrollbarHandleHoverColour
```

Field Value

Color

## scrollbarHandlePressedColour

```
public Color scrollbarHandlePressedColour
```

Field Value

Color

## shadowColour

```
public Color shadowColour
```

Field Value

Color

## subPanelColour

```
public Color subPanelColour
```

Field Value

Color

## textboxColour

```
public Color textboxColour
```

Field Value

Color

## textboxHoverColour

```
public Color textboxHoverColour
```

Field Value

Color

## textboxPressedColour

```
public Color textboxPressedColour
```

Field Value

Color

## textboxSelectedColour

```
public Color textboxSelectedColour
```

Field Value

Color

## themeName

```
public string themeName
```

Field Value

string

## toggleButtonHoverTint

```
public Color toggleButtonHoverTint
```

Field Value

Color

## toggleButtonOffColour

```
public Color toggleButtonOffColour
```

Field Value

Color

## toggleButtonOnColour

```
public Color toggleButtonOnColour
```

Field Value

Color

## toggleButtonPressedColour

```
public Color toggleButtonPressedColour
```

Field Value

Color

## Methods

### SubscribeToOnChanged(UnityAction)

```
public void SubscribeToOnChanged(UnityAction call)
```

Parameters

**call** UnityAction

# Class ThemeManager

Namespace: [PAC.Themes](#)

```
public class ThemeManager : MonoBehaviour
```

## Inheritance

[object](#) ← ThemeManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### currentTheme

```
public Theme currentTheme { get; }
```

Property Value

[Theme](#)

### themes

```
public List<Theme> themes { get; }
```

Property Value

[List](#)<[Theme](#)>

## Methods

### SetTheme(Theme)

```
public void SetTheme(Theme theme)
```

Parameters

theme [Theme](#)

## SetTheme(string)

```
public void SetTheme(string themeName)
```

Parameters

themeName [string](#)

## SubscribeToThemeChanged(UnityAction)

```
public void SubscribeToThemeChanged(UnityAction call)
```

Parameters

call [UnityAction](#)

# Enum ThemeObjectType

Namespace: [PAC.Themes](#)

```
public enum ThemeObjectType
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Background = 1

None = 0

Panel = 2

RadioButton = 5

Shadow = 4

SubPanel = 3

# Class UITheme

Namespace: [PAC.Themes](#)

```
public class UITheme : MonoBehaviour
```

## Inheritance

[object](#) ↗ ← UITheme

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

# Namespace PAC.Tilesets

## Classes

[Tile](#)

[TileOutlineManager](#)

[Tilesset](#)

[TilessetManager](#)

# Class Tile

Namespace: [PAC.Tilesets](#)

```
public class Tile
```

## Inheritance

[object](#) ← Tile

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### Tile(File, IntVector2, TileLayer[])

```
public Tile(File file, IntVector2 bottomLeft, TileLayer[] linkedTileLayers)
```

## Parameters

file [File](#)

bottomLeft [IntVector2](#)

linkedTileLayers [TileLayer](#)[]

The tile layers that the tile is on that are associated with each of the layers in the tile's file. Given in the same order as the layers in the tile's file.

## Properties

## bottomLeft

```
public IntVector2 bottomLeft { get; set; }
```

Property Value

[IntVector2](#)

## bottomRight

```
public IntVector2 bottomRight { get; set; }
```

Property Value

[IntVector2](#)

## centre

```
public Vector2 centre { get; }
```

Property Value

[Vector2](#)

## file

```
public File file { get; }
```

Property Value

[File](#)

## height

```
public int height { get; }
```

Property Value

[int](#)

rect

```
public IntRect rect { get; }
```

Property Value

[IntRect](#)

tileLayersAppearsOn

```
public TileLayer[] tileLayersAppearsOn { get; }
```

Property Value

[TileLayer\[\]](#)

topLeft

```
public IntVector2 topLeft { get; set; }
```

Property Value

[IntVector2](#)

topRight

```
public IntVector2 topRight { get; set; }
```

Property Value

[IntVector2](#)

width

```
public int width { get; }
```

Property Value

[int](#)

## Methods

### LayerInTileToTileLayer(Layer)

Takes in the layer (that must be in the tile's file) and returns the tile layer (that the tile is on) that the layer is associated with.

```
public TileLayer LayerInTileToTileLayer(Layer layer)
```

Parameters

layer [Layer](#)

Returns

[TileLayer](#)

### SubscribeToOnMoved(UnityAction<IntRect>)

```
public void SubscribeToOnMoved(UnityAction<IntRect> call)
```

## Parameters

call UnityAction<[IntRect](#)>

## TileLayerToLayerInTile(TileLayer)

Takes in the tile layer (that this tile must be on) and returns the layer it is associated with in the tile's file.

```
public Layer TileLayerToLayerInTile(TileLayer tileLayer)
```

## Parameters

tileLayer [TileLayer](#)

## Returns

[Layer](#)

# Class TileOutlineManager

Namespace: [PAC.Tilesets](#)

```
public class TileOutlineManager : MonoBehaviour
```

## Inheritance

[object](#) ↗ ← TileOutlineManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### HideShowTileOutline(Tile, bool)

```
public void HideShowTileOutline(Tile tile, bool show)
```

#### Parameters

tile [Tile](#)

show [bool](#) ↗

### HideTileOutline(Tile)

```
public void HideTileOutline(Tile tile)
```

#### Parameters

tile [Tile](#)

### RefreshTileOutlines()

```
public void RefreshTileOutlines()
```

## ShowTileOutline(Tile)

```
public void ShowTileOutline(Tile tile)
```

### Parameters

tile [Tile](#)

# Class Tileset

Namespace: [PAC.Tilesets](#)

```
public class Tileset
```

## Inheritance

[object](#) ← Tileset

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### Tileset(string)

```
public Tileset(string name)
```

#### Parameters

name [string](#)

### Tileset(string, List<File>)

```
public Tileset(string name, List<File> tiles)
```

#### Parameters

name [string](#)

`tiles` [List](#)<File>

## Properties

### Count

`public int Count { get; }`

#### Property Value

[int](#)

### tilesetName

`public string tilesetName { get; }`

#### Property Value

[string](#)

## Methods

### AddTile(File)

`public void AddTile(File tile)`

#### Parameters

`tile` [File](#)

### RemoveTile(File)

`public bool RemoveTile(File tile)`

Parameters

[tile](#) [File](#)

Returns

[bool](#) ↗

# Class TilesetManager

Namespace: [PAC.Tilesets](#)

```
public class TilesetManager : MonoBehaviour
```

## Inheritance

[object](#) ↗ ← TilesetManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Methods

### SubscribeToOnTileIconSelected(UnityAction<File>)

```
public void SubscribeToOnTileIconSelected(UnityAction<File> call)
```

## Parameters

call [UnityAction<File>](#)

# Namespace PAC.UI

## Classes

[DialogBoxManager](#)

[UIButton](#)

[UICollapser](#)

[UIColourField](#)

[UIColourFieldGroup](#)

[UIColourPicker](#)

[UIDropdown](#)

[UIDropdownChoice](#)

[UIElement](#)

[UIGridPacker](#)

[UIKeyboardShortcut](#)

[UIManager](#)

[UIModalWindow](#)

[UINumberField](#)

[UIScale](#)

[UIScrollView](#)

[UITabManager](#)

[UITextbox](#)

[UITileIcon](#)

[UIToggleButton](#)

[UIToggleGroup](#)

[UIToolButton](#)

[UITooltip](#)

[UIViewport](#)

## Enums

[CollapsedState](#)

[DropdownCloseMode](#)

[DropdownDirection](#)

[GridAlignment](#)

[ScrollDirection](#)

[UIAnchorPoint](#)

[UIElementDeselectMode](#)

[UIElementSelectMode](#)

[UITextboxAnchorPoint](#)

[ViewportSide](#)

# Enum CollapsedState

Namespace: [PAC.UI](#)

```
public enum CollapsedState
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Collapsed = 1

Uncollapsed = 0

# Class DialogBoxManager

Namespace: [PAC.UI](#)

```
public class DialogBoxManager : MonoBehaviour
```

## Inheritance

[object](#) ↗ DialogBoxManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### CloseBrushSettingsWindow()

```
public void CloseBrushSettingsWindow()
```

### CloseDialogBox(GameObject)

```
public void CloseDialogBox(GameObject dialogBox)
```

#### Parameters

**dialogBox** GameObject

### CloseDialogBox(GameObject, Action)

```
public void CloseDialogBox(GameObject dialogBox, Action onComplete)
```

#### Parameters

**dialogBox** GameObject

onComplete [Action](#)

## CloseExtendCropWindow()

```
public void CloseExtendCropWindow()
```

## CloseGridWindow()

```
public void CloseGridWindow()
```

## CloseImportPACWindow()

```
public void CloseImportPACWindow()
```

## CloseKeyboardShortcutsWindow()

```
public void CloseKeyboardShortcutsWindow()
```

## CloseLayerPropertiesWindow()

```
public void CloseLayerPropertiesWindow()
```

## CloseModalWindow(UIModalWindow)

```
public void CloseModalWindow(UIModalWindow modalWindow)
```

### Parameters

modalWindow [UIModalWindow](#)

## CloseNewFileWindow()

```
public void CloseNewFileWindow()
```

## CloseOutlineWindow()

```
public void CloseOutlineWindow()
```

## CloseReplaceColourWindow()

```
public void CloseReplaceColourWindow()
```

## CloseScaleWindow()

```
public void CloseScaleWindow()
```

## ConfirmExtendCropWindow()

```
public void ConfirmExtendCropWindow()
```

## ConfirmGridWindow()

```
public void ConfirmGridWindow()
```

## ConfirmImportPACWindow()

```
public void ConfirmImportPACWindow()
```

## ConfirmNewFileWindow()

```
public void ConfirmNewFileWindow()
```

## ConfirmOutlineWindow()

```
public void ConfirmOutlineWindow()
```

## ConfirmReplaceColourWindow()

```
public void ConfirmReplaceColourWindow()
```

## ConfirmScaleWindow()

```
public void ConfirmScaleWindow()
```

## OpenBrushSettingsWindow()

```
public void OpenBrushSettingsWindow()
```

## OpenDialogBox(GameObject)

```
public void OpenDialogBox(GameObject dialogBox)
```

### Parameters

`dialogBox` GameObject

## OpenExtendCropWindow()

```
public void OpenExtendCropWindow()
```

## OpenGridWindow()

```
public void OpenGridWindow()
```

## OpenImportPACWindow(File)

```
public void OpenImportPACWindow(File file)
```

### Parameters

file [File](#)

## OpenKeyboardShortcutsWindow()

```
public void OpenKeyboardShortcutsWindow()
```

## OpenLayerPropertiesWindow(int)

```
public void OpenLayerPropertiesWindow(int layerIndex)
```

### Parameters

layerIndex [int](#)

## OpenModalWindow()

```
public UIModalWindow OpenModalWindow()
```

Returns

[UIModalWindow](#)

## OpenModalWindow(string, string)

```
public UIModalWindow OpenModalWindow(string title, string message)
```

Parameters

**title** [string](#)

**message** [string](#)

Returns

[UIModalWindow](#)

## OpenModalWindow(string, string, string[], UnityAction[])

```
public UIModalWindow OpenModalWindow(string title, string message, string[]
buttonTexts, UnityAction[] buttonOnClicks)
```

Parameters

**title** [string](#)

**message** [string](#)

**buttonTexts** [string](#)[]

**buttonOnClicks** UnityAction[]

Returns

[UIModalWindow](#)

## OpenNewFileWindow()

```
public void OpenNewFileWindow()
```

## OpenOutlineWindow()

```
public void OpenOutlineWindow()
```

## OpenReplaceColourWindow()

```
public void OpenReplaceColourWindow()
```

## OpenScaleWindow()

```
public void OpenScaleWindow()
```

## OpenUnsavedChangesWindow(int)

```
public void OpenUnsavedChangesWindow(int fileIndex)
```

### Parameters

fileIndex [int](#)

## SetNewFileHeight(int)

```
public void SetNewFileHeight(int height)
```

### Parameters

height [int](#)

## SetNewFileWidth(int)

```
public void SetNewFileWidth(int width)
```

### Parameters

width [int](#)

## UnsavedChangesNo()

```
public void UnsavedChangesNo()
```

## UnsavedChangesYes()

```
public void UnsavedChangesYes()
```

## UpdateBrushSettingsShape()

```
public void UpdateBrushSettingsShape()
```

## UpdateImportPACPreview()

```
public void UpdateImportPACPreview()
```

# Enum DropdownCloseMode

Namespace: [PAC.UI](#)

```
public enum DropdownCloseMode
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

clickOff = 0

MouseOff = 1

# Enum DropdownDirection

Namespace: [PAC.UI](#)

```
public enum DropdownDirection
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

down = -1

up = 1

# Enum GridAlignment

Namespace: [PAC.UI](#)

```
public enum GridAlignment
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Centre = 0

Left = -1

Right = 1

# Enum ScrollDirection

Namespace: [PAC.UI](#)

```
public enum ScrollDirection
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Horizontal = 1

Vertical = 0

# Enum UIAnchorPoint

Namespace: [PAC.UI](#)

```
public enum UIAnchorPoint
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

bottomCentre = 7

bottomLeft = 6

bottomRight = 8

centre = 4

leftCentre = 3

rightCentre = 5

topCentre = 1

topLeft = 0

topRight = 2

# Class UIButton

Namespace: [PAC.UI](#)

```
public class UIButton : MonoBehaviour
```

## Inheritance

[object](#) ← UIButton

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### backgroundColour

```
public Color backgroundColour
```

Field Value

Color

### backgroundHoverColour

```
public Color backgroundHoverColour
```

Field Value

Color

### backgroundPressedColour

```
public Color backgroundPressedColour
```

Field Value

Color

## Properties

### height

```
public float height { get; set; }
```

Property Value

[float](#)

### hoverImage

```
public Sprite hoverImage { get; }
```

Property Value

Sprite

### image

```
public Sprite image { get; }
```

Property Value

Sprite

### isPressed

```
public bool isPressed { get; }
```

Property Value

bool ↴

## pressedImage

```
public Sprite pressedImage { get; }
```

Property Value

Sprite

## width

```
public float width { get; set; }
```

Property Value

float ↴

## Methods

### Press()

```
public void Press()
```

### RightClick()

```
public void RightClick()
```

### SetImages(Sprite, Sprite, Sprite)

```
public void SetImages(Sprite image, Sprite hoverImage, Sprite pressedImage)
```

Parameters

`image` Sprite

`hoverImage` Sprite

`pressedImage` Sprite

## SetSortingLayer(string)

```
public void SetSortingLayer(string sortingLayer)
```

Parameters

`sortingLayer` [string](#)

## SetSortingLayer(string, int)

```
public void SetSortingLayer(string sortingLayer, int sortingOrder)
```

Parameters

`sortingLayer` [string](#)

`sortingOrder` [int](#)

## SetSortingOrder(int)

```
public void SetSortingOrder(int sortingOrder)
```

Parameters

`sortingOrder` [int](#)

## SetText(string)

```
public void SetText(string text)
```

### Parameters

**text** [string](#)

## SubscribeToClick(UnityAction)

```
public void SubscribeToClick(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToHover(UnityAction)

```
public void SubscribeToHover(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToIdle(UnityAction)

```
public void SubscribeToIdle(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToLeftClick(UnityAction)

```
public void SubscribeToLeftClick(UnityAction call)
```

### Parameters

**call** UnityAction

## SubscribeToRightClick(UnityAction)

```
public void SubscribeToRightClick(UnityAction call)
```

### Parameters

**call** UnityAction

## UpdateDisplay()

```
public void UpdateDisplay()
```

# Class UICollapser

Namespace: [PAC.UI](#)

```
public class UICollapser : MonoBehaviour
```

## Inheritance

[object](#) ← UICollapser

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#).

## Methods

### Collapse()

```
public void Collapse()
```

### SetCollapsed(CollapsedState)

```
public void SetCollapsed(CollapsedState collapsedState)
```

#### Parameters

collapsedState [CollapsedState](#)

### Uncollapse()

```
public void Uncollapse()
```

# Class UIColourField

Namespace: [PAC.UI](#)

```
public class UIColourField : MonoBehaviour
```

## Inheritance

[object](#) ← UIColourField

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### colour

```
public Color colour { get; }
```

Property Value

Color

### colourPickerOpen

```
public bool colourPickerOpen { get; }
```

Property Value

[bool](#)

### outlineThickness

```
public float outlineThickness { get; }
```

Property Value

[float ↻](#)

## Methods

### CloseColourPicker()

```
public void CloseColourPicker()
```

### OpenColourPicker()

```
public void OpenColourPicker()
```

### SetColour(Color)

```
public void SetColour(Color colour)
```

#### Parameters

`colour` Color

### SubscribeToColourChange(UnityAction)

```
public void SubscribeToColourChange(UnityAction call)
```

#### Parameters

`call` UnityAction

### SubscribeToColourPickerClose(UnityAction)

```
public void SubscribeToColourPickerClose(UnityAction call)
```

Parameters

**call** UnityAction

## SubscribeToColourPickerOpen(UnityAction)

```
public void SubscribeToColourPickerOpen(UnityAction call)
```

Parameters

**call** UnityAction

# Class UIColourFieldGroup

Namespace: [PAC.UI](#)

```
public class UIColourFieldGroup : MonoBehaviour
```

## Inheritance

[object](#) ← UIColourFieldGroup

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### Count

```
public int Count { get; }
```

### Property Value

[int](#)

### colourFields

```
public List<UIColourField> colourFields { get; }
```

### Property Value

[List](#)<[UIColourField](#)>

### currentOpenColourField

```
public UIColourField currentOpenColourField { get; }
```

Property Value

[UIColourField](#)

## Methods

### Add(UIColourField)

```
public bool Add(UIColourField colourField)
```

Parameters

colourField [UIColourField](#)

Returns

[bool](#)

### Clear()

```
public void Clear()
```

### Contains(UIColourField)

```
public bool Contains(UIColourField colourField)
```

Parameters

colourField [UIColourField](#)

Returns

[bool](#)

## DestroyColourFields()

```
public void DestroyColourFields()
```

## Opened(UIColourField)

```
public void Opened(UIColourField openedColourField)
```

### Parameters

openedColourField [UIColourField](#)

## Remove(UIColourField)

```
public bool Remove(UIColourField colourField)
```

### Parameters

colourField [UIColourField](#)

### Returns

[bool](#)

## SubscribeToColourFieldOpen(UnityAction)

```
public void SubscribeToColourFieldOpen(UnityAction call)
```

### Parameters

call UnityAction

# Class UIColourPicker

Namespace: [PAC.UI](#)

```
public class UIColourPicker : MonoBehaviour
```

## Inheritance

[object](#) ← UIColourPicker

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### colourPreview

```
public ColourPreview colourPreview
```

Field Value

[ColourPreview](#)

## Properties

### colour

```
public Color colour { get; }
```

Property Value

Color

## Methods

## Close()

```
public void Close()
```

## SelectDeselectEyeDropper()

```
public void SelectDeselectEyeDropper()
```

## SetColour(Color)

```
public void SetColour(Color colour)
```

### Parameters

`colour` Color

## SubscribeToClose(UnityAction)

```
public void SubscribeToClose(UnityAction call)
```

### Parameters

`call` UnityAction

## SubscribeToColourChange(UnityAction)

```
public void SubscribeToColourChange(UnityAction call)
```

### Parameters

`call` UnityAction

## UpdateColour(Color)

```
public void UpdateColour(Color colour)
```

### Parameters

colour Color

# Class UIDropdown

Namespace: [PAC.UI](#)

```
public class UIDropdown : MonoBehaviour
```

## Inheritance

[object](#) ← UIDropdown

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### openingButton

```
public GameObject openingButton
```

Field Value

GameObject

### parentDropdown

```
public UIDropdown parentDropdown
```

Field Value

[UIDropdown](#)

## Properties

### isRootDropdown

```
public bool isRootDropdown { get; }
```

Property Value

[bool](#)

open

```
public bool open { get; }
```

Property Value

[bool](#)

rootDropdown

```
public UIDropdown rootDropdown { get; }
```

Property Value

[UIDropdown](#)

Methods

Close()

```
public void Close()
```

CloseRoot()

Closes the highest-level dropdown containing this one.

```
public void CloseRoot()
```

## FullyOpen()

Opens this dropdown and all child dropdowns, and all their child dropdowns, etc.

```
public void FullyOpen()
```

## Initialise()

```
public void Initialise()
```

## MouseOff()

```
public bool MouseOff()
```

Returns

bool

## Open()

```
public void Open()
```

## SetOpen(bool)

```
public void SetOpen(bool open)
```

Parameters

open bool

## SetOpenEditor(bool)

```
public void SetOpenEditor(bool open)
```

Parameters

open [bool](#)

## ToggleOpen()

```
public void ToggleOpen()
```

# Class UIDropdownChoice

Namespace: [PAC.UI](#)

```
public class UIDropdownChoice : MonoBehaviour
```

## Inheritance

[object](#) ← UIDropdownChoice

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### direction

```
public DropdownDirection direction
```

Field Value

[DropdownDirection](#)

### options

```
public List<string> options
```

Field Value

[List<string>](#)

### selectedOptionIndex

```
public int selectedOptionIndex
```

Field Value

[int](#)

## Properties

### selectedOption

```
public string selectedOption { get; }
```

Property Value

[string](#)

## Methods

### Select(string)

```
public bool Select(string option)
```

Parameters

option [string](#)

Returns

[bool](#)

### SetUpDropdown()

```
public void SetUpDropdown()
```

### SubscribeToOptionChanged(UnityAction)

```
public void SubscribeToOptionChanged(UnityAction call)
```

## Parameters

**call** UnityAction

# Class UIElement

Namespace: [PAC.UI](#)

```
public class UIElement : MonoBehaviour
```

## Inheritance

[object](#) ↴ ← UIElement

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### viewport

```
public UIPortrait viewport
```

#### Field Value

[UIViewport](#)

## Properties

### elementName

```
public string elementName { get; }
```

#### Property Value

[string](#) ↴

# Enum UIElementDeselectMode

Namespace: [PAC.UI](#)

```
public enum UIElementDeselectMode
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Manual = 0

OnInputTargetUntargeted = 1

# Enum UIElementSelectMode

Namespace: [PAC.UI](#)

```
public enum UIElementSelectMode
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Manual = 0

OnInputTargetTargeted = 1

# Class UIGridPacker

Namespace: [PAC.UI](#)

```
public class UIGridPacker : MonoBehaviour
```

## Inheritance

[object](#) ← UIGridPacker

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### Repack()

```
public void Repack()
```

# Class UIKeyboardShortcut

Namespace: [PAC.UI](#)

```
public class UIKeyboardShortcut : MonoBehaviour
```

## Inheritance

[object](#) ← UIKeyboardShortcut

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### actionName

```
public string actionName
```

#### Field Value

[string](#)

## Properties

### shortcut

```
public KeyboardShortcut shortcut { get; set; }
```

#### Property Value

[KeyboardShortcut](#)

## Methods

## SubscribeToOnShortcutSet(UnityAction<KeyboardShortcut>)

```
public void SubscribeToOnShortcutSet(UnityAction<KeyboardShortcut> call)
```

### Parameters

call UnityAction<[KeyboardShortcut](#)>

# Class UIManager

Namespace: [PAC.UI](#)

```
public class UIManager : MonoBehaviour
```

## Inheritance

[object](#) ← UIManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### selectedUIElement

```
public UIElement selectedUIElement { get; }
```

Property Value

[UIElement](#)

## Methods

### CanTargetInputTarget(InputTarget)

```
public bool CanTargetInputTarget(InputTarget inputTarget)
```

Parameters

inputTarget [InputTarget](#)

Returns

[bool](#)

## TryTarget(UIElement)

```
public bool TryTarget(UIElement uiElement)
```

Parameters

uiElement [UIElement](#)

Returns

[bool](#)

## TryUntarget(UIElement)

```
public bool TryUntarget(UIElement uiElement)
```

Parameters

uiElement [UIElement](#)

Returns

[bool](#)

# Class UIModalWindow

Namespace: [PAC.UI](#)

```
public class UIModalWindow : MonoBehaviour
```

## Inheritance

[object](#) ↗ UIModalWindow

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### AddButton(string, UnityAction)

```
public UIModalWindow AddButton(string text, UnityAction onClick)
```

#### Parameters

text [string](#) ↗

onClick [UnityAction](#)

#### Returns

[UIModalWindow](#)

### AddCloseButton(string)

```
public UIModalWindow AddCloseButton(string text)
```

#### Parameters

text [string](#) ↗

Returns

[UIModalWindow](#)

**Close()**

```
public void Close()
```

**SetMessage(string)**

```
public UIPeripheralWindow SetMessage(string message)
```

Parameters

message [string](#)

Returns

[UIPeripheralWindow](#)

**SetTitle(string)**

```
public UIPeripheralWindow SetTitle(string title)
```

Parameters

title [string](#)

Returns

[UIPeripheralWindow](#)

# Class UINumberField

Namespace: [PAC.UI](#)

```
public class UINumberField : MonoBehaviour
```

## Inheritance

[object](#) ← UINumberField

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### max

```
public float max { get; set; }
```

Property Value

[float](#)

### min

```
public float min { get; set; }
```

Property Value

[float](#)

### value

```
public float value { get; set; }
```

## Property Value

`float` ↗

## Methods

### AddNumOfIncrements(int)

```
public void AddNumOfIncrements(int numOfIncrements)
```

#### Parameters

`numOfIncrements` `int` ↗

### Decrement()

```
public void Decrement()
```

### Increment()

```
public void Increment()
```

### SubscribeToValueChanged(UnityAction)

```
public void SubscribeToValueChanged(UnityAction call)
```

#### Parameters

`call` UnityAction

# Class UIScale

Namespace: [PAC.UI](#)

```
public class UIScale : MonoBehaviour
```

## Inheritance

[object](#) ← UIScale

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### backgroundColour

```
public Color backgroundColour
```

Field Value

Color

### backgroundHoverColour

```
public Color backgroundHoverColour
```

Field Value

Color

### backgroundPressedColour

```
public Color backgroundPressedColour
```

Field Value

Color

height

```
public float height
```

Field Value

[float](#)

width

```
public float width
```

Field Value

[float](#)

Properties

decimalPlaces

```
public int decimalPlaces { get; }
```

Property Value

[int](#)

value

```
public float value { get; }
```

Property Value

[float](#)

## Methods

### SetValue(float)

```
public bool SetValue(float value)
```

Parameters

[value](#) [float](#)

Returns

[bool](#)

### SetValueNoNotify(float)

```
public bool SetValueNoNotify(float value)
```

Parameters

[value](#) [float](#)

Returns

[bool](#)

### SubscribeToValueChange(UnityAction)

```
public void SubscribeToValueChange(UnityAction call)
```

Parameters

call UnityAction

## UpdateDisplay()

```
public void UpdateDisplay()
```

# Class UIScrollbar

Namespace: [PAC.UI](#)

```
public class UIScrollbar : MonoBehaviour
```

## Inheritance

[Object](#) ↗ ← UIScrollbar

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### backgroundColour

```
public Color backgroundColour
```

Field Value

Color

### backgroundHoverColour

```
public Color backgroundHoverColour
```

Field Value

Color

### backgroundPressedColour

```
public Color backgroundPressedColour
```

Field Value

Color

## handleColour

```
public Color handleColour
```

Field Value

Color

## handleHoverColour

```
public Color handleHoverColour
```

Field Value

Color

## handlePressedColour

```
public Color handlePressedColour
```

Field Value

Color

## height

```
public float height
```

Field Value

[float](#)

## width

`public float width`

### Field Value

[float](#)

## Properties

### scrollAmount

`public float scrollAmount { get; set; }`

### Property Value

[float](#)

## Methods

### GetScrollAmount()

`public float GetScrollAmount()`

### Returns

[float](#)

### SetScrollAmount(float)

`public void SetScrollAmount(float scrollAmount)`

## Parameters

scrollAmount [float](#)

## UpdateDisplay()

`public void UpdateDisplay()`

# Class UITabManager

Namespace: [PAC.UI](#)

```
public class UITabManager : MonoBehaviour
```

## Inheritance

[Object](#) ↗ ← UITabManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### AddTab(GameObject)

```
public void AddTab(GameObject tab)
```

#### Parameters

tab GameObject

### SelectTab(int)

```
public void SelectTab(int tabIndex)
```

#### Parameters

tabIndex [int](#) ↗

### SelectTabEditor(int)

```
public void SelectTabEditor(int tabIndex)
```

## Parameters

tabIndex [int](#)

# Class UITextbox

Namespace: [PAC.UI](#)

```
public class UITextbox : MonoBehaviour
```

## Inheritance

[object](#) ← UITextbox

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### allowLetters

```
public bool allowLetters
```

Field Value

[bool](#)

### allowNumbers

```
public bool allowNumbers
```

Field Value

[bool](#)

### allowPunctuation

```
public bool allowPunctuation
```

Field Value

[bool](#)

## allowSpaces

`public bool allowSpaces`

Field Value

[bool](#)

## anchorPoint

`public UITextboxAnchorPoint anchorPoint`

Field Value

[UITextboxAnchorPoint](#)

## backgroundColour

`public Color backgroundColour`

Field Value

Color

## backgroundHoverColour

`public Color backgroundHoverColour`

Field Value

Color

## backgroundPressedColour

```
public Color backgroundPressedColour
```

Field Value

Color

## backgroundSelectedColour

```
public Color backgroundSelectedColour
```

Field Value

Color

## height

```
public float height
```

Field Value

[float](#)

## width

```
public float width
```

Field Value

[float](#)

# Properties

## prefix

```
public string prefix { get; }
```

Property Value

string ↗

## suffix

```
public string suffix { get; }
```

Property Value

string ↗

## text

```
public string text { get; }
```

Property Value

string ↗

# Methods

## SetPrefix(string)

```
public void SetPrefix(string prefix)
```

Parameters

prefix [string](#)

## SetSuffix(string)

```
public void SetSuffix(string suffix)
```

Parameters

suffix [string](#)

## SetText(string)

```
public void SetText(string text)
```

Parameters

text [string](#)

## SubscribeToFinishEvent(UnityAction)

```
public void SubscribeToFinishEvent(UnityAction call)
```

Parameters

call UnityAction

## SubscribeToInputEvent(UnityAction)

```
public void SubscribeToInputEvent(UnityAction call)
```

Parameters

call UnityAction

## UpdateAnchor()

```
public void UpdateAnchor()
```

## UpdateDisplay()

```
public void UpdateDisplay()
```

# Enum UITextboxAnchorPoint

Namespace: [PAC.UI](#)

```
public enum UITextboxAnchorPoint
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Centre = 4

Left = 3

Right = 5

# Class UITileIcon

Namespace: [PAC.UI](#)

```
public class UITileIcon : MonoBehaviour
```

## Inheritance

[object](#) ↗ ← UITileIcon

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### file

```
public File file { get; }
```

Property Value

[File](#)

### height

```
public float height { get; }
```

Property Value

[float](#) ↗

### width

```
public float width { get; }
```

Property Value

[float](#)

## Methods

### SetFile(File)

```
public void SetFile(File file)
```

Parameters

[file](#) [File](#)

### SubscribeToOnClick(UnityAction)

```
public void SubscribeToOnClick(UnityAction call)
```

Parameters

[call](#) [UnityAction](#)

### SubscribeToOnLeftClick(UnityAction)

```
public void SubscribeToOnLeftClick(UnityAction call)
```

Parameters

[call](#) [UnityAction](#)

### SubscribeToOnRightClick(UnityAction)

```
public void SubscribeToOnRightClick(UnityAction call)
```

## Parameters

**call** UnityAction

# Class UIToggleButton

Namespace: [PAC.UI](#)

```
public class UIToggleButton : MonoBehaviour
```

## Inheritance

[object](#) ← UIToggleButton

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### height

```
public float height
```

Field Value

[float](#)

### hoverBackgroundTint

```
public Color hoverBackgroundTint
```

Field Value

Color

### width

```
public float width
```

Field Value

[float](#)

## Properties

### hoverImage

```
public Sprite hoverImage { get; }
```

Property Value

Sprite

### inToggleGroup

```
public bool inToggleGroup { get; }
```

Property Value

[bool](#)

### offBackgroundColour

```
public Color offBackgroundColour { get; set; }
```

Property Value

Color

### offImage

```
public Sprite offImage { get; }
```

Property Value

Sprite

on

```
public bool on { get; }
```

Property Value

bool ↗

onBackgroundColour

```
public Color onBackgroundColour { get; set; }
```

Property Value

Color

onImage

```
public Sprite onImage { get; }
```

Property Value

Sprite

pressedBackgroundColour

```
public Color pressedBackgroundColour { get; set; }
```

Property Value

Color

## pressedImage

```
public Sprite pressedImage { get; }
```

Property Value

Sprite

## toggleGroup

```
public UIToggleGroup toggleGroup { get; }
```

Property Value

[UIToggleGroup](#)

## toggleName

```
public string toggleName { get; set; }
```

Property Value

[string](#) ↗

## Methods

### JoinToggleGroup(UIToggleGroup)

```
public void JoinToggleGroup(UIToggleGroup toggleGroup)
```

Parameters

toggleGroup [UIToggleGroup](#)

## LeaveToggleGroup()

```
public void LeaveToggleGroup()
```

## Press()

```
public void Press()
```

## RightClick()

```
public void RightClick()
```

## SetImages(Sprite, Sprite, Sprite, Sprite)

```
public void SetImages(Sprite offImage, Sprite onImage, Sprite hoverImage,  
Sprite pressedImage)
```

### Parameters

offImage Sprite

onImage Sprite

hoverImage Sprite

pressedImage Sprite

## SetOnOff(bool)

```
public void SetOnOff(bool on)
```

Parameters

on bool ↴

## SetText(string)

```
public void SetText(string text)
```

Parameters

text string ↴

## SetTextAlignment(TextAnchor)

```
public void SetTextAlignment(TextAnchor textAlignment)
```

Parameters

textAlignment TextAnchor

## SubscribeToHover(UnityAction)

```
public void SubscribeToHover(UnityAction call)
```

Parameters

call UnityAction

## SubscribeToLeftClick(UnityAction)

```
public void SubscribeToLeftClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToRightClick(UnityAction)

```
public void SubscribeToRightClick(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToTurnOff(UnityAction)

```
public void SubscribeToTurnOff(UnityAction call)
```

Parameters

`call` UnityAction

## SubscribeToTurnOn(UnityAction)

```
public void SubscribeToTurnOn(UnityAction call)
```

Parameters

`call` UnityAction

## UpdateDisplay()

```
public void UpdateDisplay()
```

# Class UIToggleGroup

Namespace: [PAC.UI](#)

```
public class UIToggleGroup : MonoBehaviour
```

## Inheritance

[object](#) ← UIToggleGroup

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### Count

```
public int Count { get; }
```

### Property Value

[int](#)

### canSelectMultiple

```
public bool canSelectMultiple { get; }
```

### Property Value

[bool](#)

### canSelectNone

```
public bool canSelectNone { get; }
```

Property Value

[bool](#)

## currentToggle

```
public UIToggleButton currentToggle { get; }
```

Property Value

[UIToggleButton](#)

## currentToggleIndex

```
public int currentToggleIndex { get; }
```

Property Value

[int](#)

## hasSelectedToggle

```
public bool hasSelectedToggle { get; }
```

Property Value

[bool](#)

## selectedIndices

```
public int[] selectedIndices { get; }
```

Property Value

[int](#)[]

## selectedToggles

```
public UIToggleButton[] selectedToggles { get; }
```

Property Value

[UIToggleButton](#)[]

## swapClickAndCtrlClick

```
public bool swapClickAndCtrlClick { get; }
```

Property Value

[bool](#)

## toggles

```
public List<UIToggleButton> toggles { get; }
```

Property Value

[List](#)<[UIToggleButton](#)>

## Methods

### Add(UIToggleButton)

```
public bool Add(UIToggleButton toggle)
```

Parameters

`toggle` [UIToggleButton](#)

Returns

[bool](#)

## Clear()

```
public void Clear()
```

## Contains(UIToggleButton)

```
public bool Contains(UIToggleButton toggle)
```

Parameters

`toggle` [UIToggleButton](#)

Returns

[bool](#)

## CtrlPress(UIToggleButton)

```
public bool CtrlPress(UIToggleButton toggle)
```

Parameters

`toggle` [UIToggleButton](#)

Returns

[bool](#)

## CtrlPress(int)

```
public bool CtrlPress(int index)
```

Parameters

index [int](#)

Returns

[bool](#)

## DestroyToggles()

```
public void DestroyToggles()
```

## Press(UIToggleButton)

```
public bool Press(UIToggleButton toggle)
```

Parameters

toggle [UIToggleButton](#)

Returns

[bool](#)

## Press(int)

```
public bool Press(int index)
```

Parameters

index [int](#)

Returns

[bool](#)

## PressForceEvent(UIToggleButton)

```
public bool PressForceEvent(UIToggleButton toggle)
```

Parameters

[toggle](#) [UIToggleButton](#)

Returns

[bool](#)

## PressForceEvent(int)

```
public bool PressForceEvent(int index)
```

Parameters

[index](#) [int](#)

Returns

[bool](#)

## PressOrCtrlPress(UIToggleButton, bool)

```
public bool PressOrCtrlPress(UIToggleButton toggle, bool ctrlClick)
```

Parameters

`toggle` [UIToggleButton](#)

`ctrlClick` [bool](#) ↗

Returns

[bool](#) ↗

## Remove(UIToggleButton)

`public bool Remove(UIToggleButton toggle)`

Parameters

`toggle` [UIToggleButton](#)

Returns

[bool](#) ↗

## SubscribeToSelectedToggleChange(UnityAction)

`public void SubscribeToSelectedToggleChange(UnityAction call)`

Parameters

`call` UnityAction

# Class UIToolButton

Namespace: [PAC.UI](#)

```
public class UIToolButton : MonoBehaviour
```

## Inheritance

[object](#) ← UIToolButton

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Properties

### buttons

```
public UIButton[] buttons { get; }
```

Property Value

[UIButton\[\]](#)

### currentButton

```
public UIButton currentButton { get; }
```

Property Value

[UIButton](#)

# Class UITooltip

Namespace: [PAC.UI](#)

```
public class UITooltip : MonoBehaviour
```

## Inheritance

[object](#) ← UITooltip

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### padding

```
public Vector2 padding
```

#### Field Value

Vector2

## Properties

### globalHeight

```
public float globalHeight { get; }
```

#### Property Value

[float](#)

### globalWidth

```
public float globalWidth { get; }
```

Property Value

[float](#) ↗

## localHeight

```
public float localHeight { get; }
```

Property Value

[float](#) ↗

## localWidth

```
public float localWidth { get; }
```

Property Value

[float](#) ↗

## text

```
public string text { get; set; }
```

Property Value

[string](#) ↗

## Methods

### Awake()

```
public void Awake()
```

## GoesOffBottomOfScreen()

```
public bool GoesOffBottomOfScreen()
```

Returns

[bool](#)

## GoesOffLeftOfScreen()

```
public bool GoesOffLeftOfScreen()
```

Returns

[bool](#)

## GoesOffRightOfScreen()

```
public bool GoesOffRightOfScreen()
```

Returns

[bool](#)

## GoesOffScreen()

```
public bool GoesOffScreen()
```

Returns

[bool](#)

## GoesOffTopOfScreen()

```
public bool GoesOffTopOfScreen()
```

Returns

[bool](#)

## SetText(string)

```
public void SetText(string text)
```

Parameters

text [string](#)

# Class UIViewport

Namespace: [PAC.UI](#)

```
public class UIViewport : MonoBehaviour
```

## Inheritance

[object](#) ← UIViewport

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

### boundScrollToContents

```
public bool boundScrollToContents
```

Field Value

[bool](#)

### maxScrollAmount

```
public float maxScrollAmount
```

Field Value

[float](#)

### maxScrollEnabled

```
public bool maxScrollEnabled
```

Field Value

[bool ↗](#)

## minScrollAmount

`public float minScrollAmount`

Field Value

[float ↗](#)

## minScrollEnabled

`public bool minScrollEnabled`

Field Value

[bool ↗](#)

## scrollDirection

`public ScrollDirection scrollDirection`

Field Value

[ScrollDirection](#)

## Properties

### collider

`public BoxCollider2D collider { get; }`

Property Value

BoxCollider2D

## defaultScrollSide

```
public ViewportSide defaultScrollSide { get; }
```

Property Value

[ViewportSide](#)

## rectTransform

```
public RectTransform rectTransform { get; }
```

Property Value

RectTransform

## scrollAmount

```
public float scrollAmount { get; }
```

Property Value

[float](#)

## scrollingArea

```
public Transform scrollingArea { get; }
```

Property Value

## Methods

### AddScrollAmount(float)

```
public void AddScrollAmount(float scrollAmount)
```

#### Parameters

scrollAmount [float](#)

### GetScrollMinMaxX()

Get the min and max x value of the space that the objects in this viewport's scroll area occupy. (Scaled to the scale of the viewport object.)

```
public Vector2 GetScrollMinMaxX()
```

#### Returns

Vector2

(min x, max x)

### GetScrollMinMaxY()

Get the min and max y value of the space that the objects in this viewport's scroll area occupy. (Scaled to the scale of the viewport object.)

```
public Vector2 GetScrollMinMaxY()
```

#### Returns

Vector2

(min y, max y)

## RefreshViewport()

```
public void RefreshViewport()
```

## SetScrollAmount(float)

```
public void SetScrollAmount(float scrollAmount)
```

### Parameters

scrollAmount [float](#)

## SubscribeToRefresh(UnityAction)

```
public void SubscribeToRefresh(UnityAction call)
```

### Parameters

call UnityAction

# Enum ViewportSide

Namespace: [PAC.UI](#)

```
public enum ViewportSide
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Centre = 0

Negative = -1

Positive = 1

# Namespace PAC.UndoRedo

## Classes

[UndoRedoManager](#)

[UndoRedoState](#)

## Enums

[UndoRedoAction](#)

# Enum UndoRedoAction

Namespace: [PAC.UndoRedo](#)

```
public enum UndoRedoAction
```

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Fields

Draw = 1

None = -1

ReorderLayers = 2

Undefined = 0

# Class UndoRedoManager

Namespace: [PAC.UndoRedo](#)

```
public class UndoRedoManager : MonoBehaviour
```

## Inheritance

[object](#) ↗ UndoRedoManager

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Methods

### AddUndoState(UndoRedoState, int)

```
public void AddUndoState(UndoRedoState undoState, int fileIndex)
```

#### Parameters

undoState [UndoRedoState](#)

fileIndex [int](#) ↗

### SubscribeToRedo(UnityAction)

```
public void SubscribeToRedo(UnityAction call)
```

#### Parameters

call UnityAction

### SubscribeToUndo(UnityAction)

```
public void SubscribeToUndo(UnityAction call)
```

Parameters

**call** UnityAction

## SubscribeToUndoOrRedo(UnityAction)

```
public void SubscribeToUndoOrRedo(UnityAction call)
```

Parameters

**call** UnityAction

# Class UndoRedoState

Namespace: [PAC.UndoRedo](#)

```
public class UndoRedoState
```

## Inheritance

[object](#) ← UndoRedoState

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)

## Constructors

### UndoRedoState(UndoRedoAction, Layer, int)

```
public UndoRedoState(UndoRedoAction action, Layer layer, int layerIndex)
```

#### Parameters

action [UndoRedoAction](#)

layer [Layer](#)

layerIndex [int](#)

### UndoRedoState(UndoRedoAction, Layer[], int[])

```
public UndoRedoState(UndoRedoAction action, Layer[] affectedLayers,  
int[] affectedLayersIndices)
```

## Parameters

action [UndoRedoAction](#)

affectedLayers [Layer\[\]](#)

affectedLayersIndices [int\[\]](#)[]

## Properties

### action

```
public UndoRedoAction action { get; }
```

Property Value

[UndoRedoAction](#)

### affectedLayers

```
public Layer[] affectedLayers { get; }
```

Property Value

[Layer\[\]](#)

### affectedLayersIndices

```
public int[] affectedLayersIndices { get; }
```

Property Value

[int\[\]](#)[]

# Namespace System.Runtime.CompilerServices

## Classes

[IsExternalInit](#)

# Class IsExternalInit

Namespace: [System.Runtime.CompilerServices](#)

```
public class IsExternalInit
```

## Inheritance

[object](#) ← IsExternalInit

## Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) ,  
[object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) ,  
[object.ToString\(\)](#)

## Extension Methods

[ISetComparableExtensions.IsProperSubsetOf<T1, T2>\(T1, T2\)](#) ,  
[ISetComparableExtensions.IsProperSupersetOf<T1, T2>\(T1, T2\)](#)