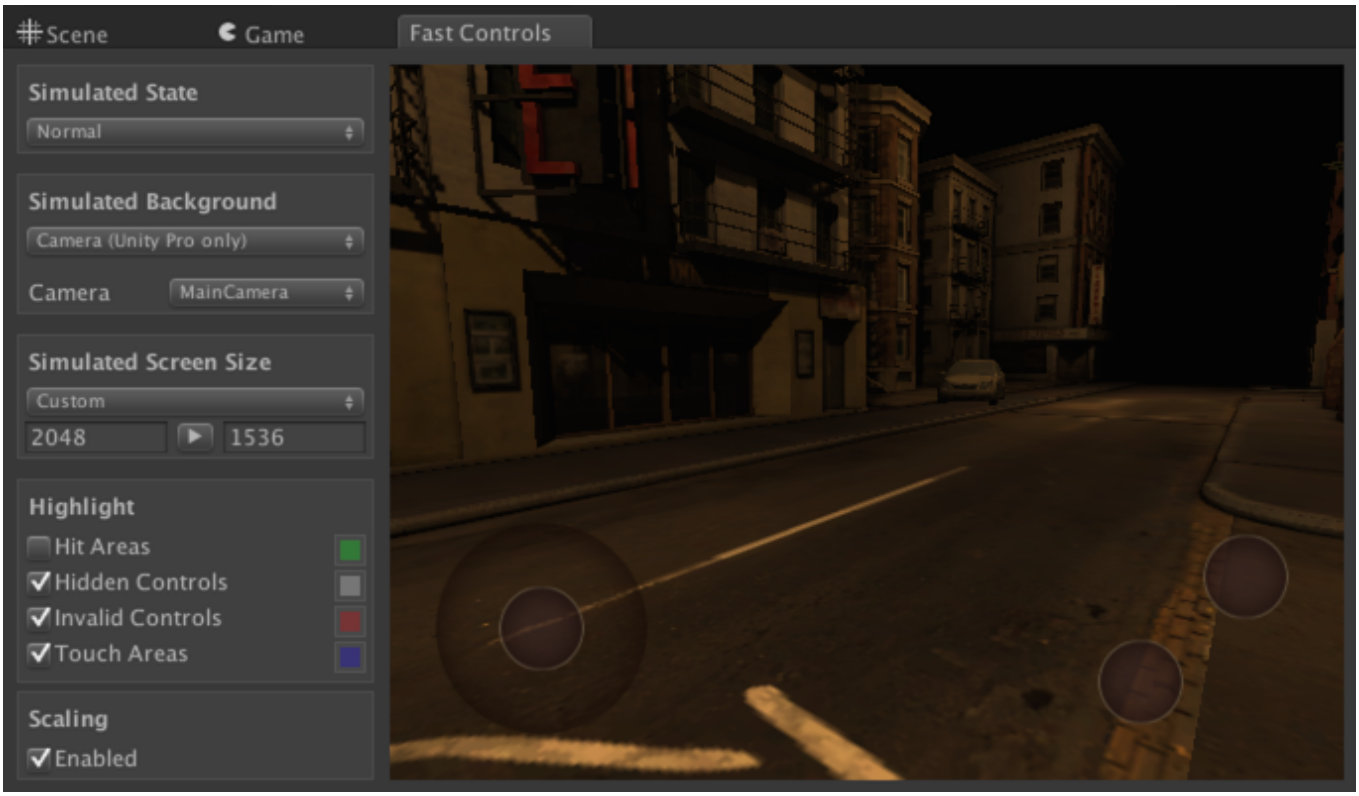
Description

Fast Controls allows the game to render thumb sticks and buttons super fast with a single draw call. Touch controls can be easily created with the visual editor and the required texture atlas will be automatically generated by simply dragging and dropping the texture assets to the designer. Fast Controls is designed for touch devices and it supports touching multiple controls at the same time.

Adding a Fast Controls renderer

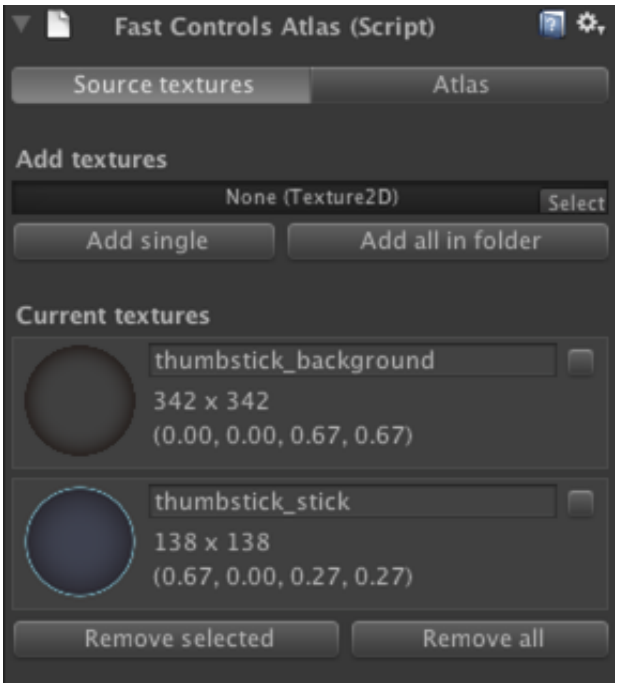
The first thing to do is create and an empty GameObject and add the FastControlsRenderer component on it. By selecting the GameObject, a new window called FastControls is opened. This window is the preview of your Fast Controls renderer instance. The preview window has a small vertical toolbar which allows you to change the simulated, background and screen size while designing.



Adding the texture assets

The GameObject created above will also contain a component called FastControlsAtlas which is the automatic texture atlas generation tool. To create the required texture atlas you should simply drag and drop your touch control texture asset to the texture slot and click Add single to add it to the atlas. You may also add all textures inside the same folder by clicking the Add

all in folder instead of dragging them all one by one. You may anytime add more items or remove items and the atlas will update automatically. After you have added your texture assets you can start adding the actual touch controls.

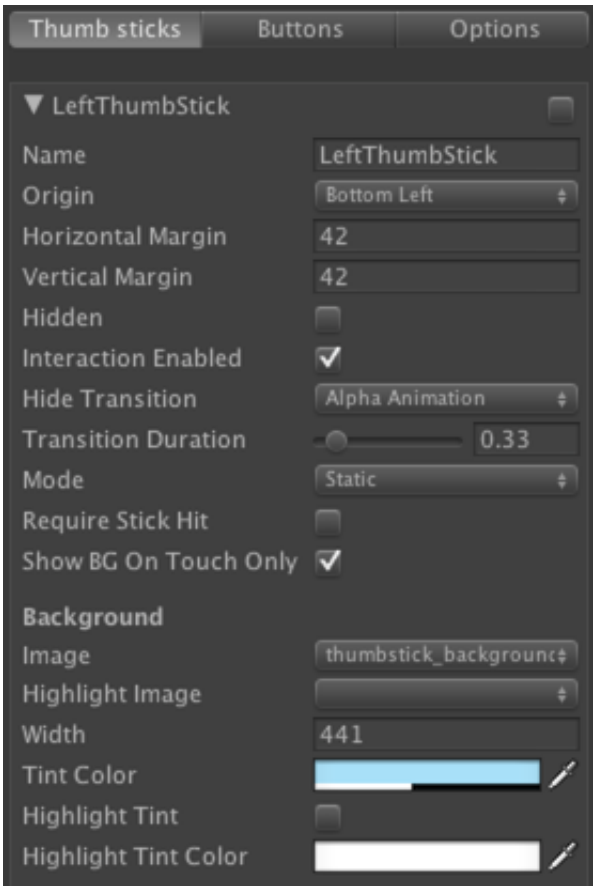


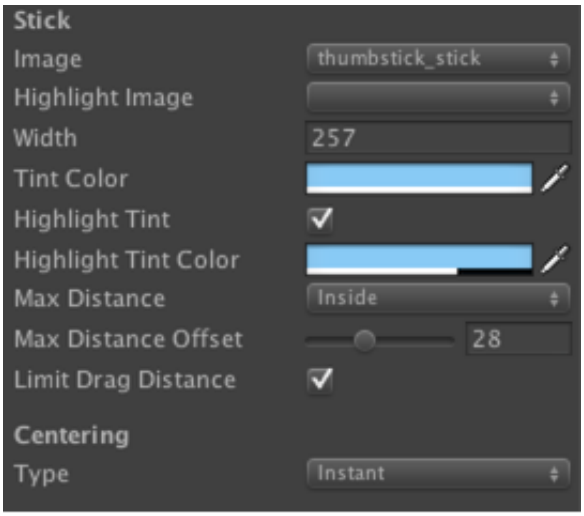
Thumb sticks

Thumb stick is a control similar to joystick which can be used for example to move a game character or to aim a weapon. There are two different types of thumb sticks, static thumb sticks which always stay at the same location and touch area thumb sticks which will automatically hide when finger is released and reapper to the new touch position if the touch is insided the defined touch area. Thumb sticks require two textures, a background texture and a stick texture. Like any other control, there is no limit how many thumb sticks you may add to your renderer. You may also modify all control properties at runtime, for example hide the control or change it's size and so on.

Adding a thumb stick

To add a new thumb stick, make sure that thumb sticks tab on your Fast Controls renderer is selected. By clicking the Add button a new blank thumb stick with default properties will be added to the designer. There are many options to set however you should at least define the background and stick textures using the background/stick image drop down menus. All control options are named in a self-explanatory way and they also contain tooltips so only some of them are covered in this manual. To see the tooltips, just move your mouse cursor on top of the option label and more info will appear.





Listening for thumb stick changes

When a thumb stick is moved it's magnitude and direction will change. You can listen for thumb stick changes either by polling the control values or by using event handlers that will

get only triggered when an actual change happens. An event handler can be defined on any instanciated GameObject, either in editor or runtime. There is no limit how many event handlers there are for the same control.

The event listener GameObject should have a script that implements a method that conforms to the ThumbStickEventHandler delegate method.

```
public delegate void ThumbStickEventHandler(FastControls.ThumbStickEventArgs args)
```

After the event handler method is defined the GameObject may simply be dragged and dropped to the Fast Controls designer's event slot and the desired method can be selected from the dropdown menu.

Example thumb stick event handlers

```
public void ThumbStickDown(FastControls.ThumbStickEventArgs args) {
    Debug.Log ("Thumb stick down, touch position " + args.touchPosition);
}

public void ThumbStickDragged(FastControls.ThumbStickEventArgs args) {
    Debug.Log ("Thumb stick dragged, touch position " + args.touchPosition + ", move
ment delta " + args.touchDeltaPosition);
}

public void ThumbStickUp(FastControls.ThumbStickEventArgs args) {
    Debug.Log ("Thumb stick up, touch position " + args.touchPosition);
}

public void ThumbStickValuesChanged(FastControls.ThumbStickEventArgs args) {
    Debug.Log ("Thumb stick values changed, direction is " + args.eventSender.direct
ion + " and magnitude " + args.eventSender.magnitude);
}
```

But I want to set my event handlers at runtime!

Instead of dragging and dropping, you can also set your event handlers in a script at runtime by first getting a reference to your thumb stick and then subscribing to some or all of the following events. If you subscribe at runtime you are responsible to unsubscribe when destroying the object to avoid leaking memory. To support enabling/disabling of GameObjects it is recommended to do the subscribing in OnEnable method and the unsubscribing in OnDisable method.

```
public event ThumbStickEventHandler OnStickDown
public event ThumbStickEventHandler OnStickDragged
public event ThumbStickEventHandler OnStickUp
```

```
public event ThumbStickEventHandler OnValuesChanged
```

Runtime thumb stick event subscribing example

```
void OnEnable() {
    FastControls.ThumbStick leftThumbStick = myFastControlsInstance.GetThumbStickByN
ame("LeftThumbStick");

    if(leftThumbStick != null) {
        leftThumbStick.OnStickDown += ThumbStickDown;
        leftThumbStick.OnStickDragged += ThumbStickDragged;
        leftThumbStick.OnStickUp += ThumbStickUp;
        leftThumbStick.OnValuesChanged += ThumbStickValuesChanged;
    }
}

void OnDisable() {
    FastControls.RemoveAllEventHandlersForObject(this);
}
```

Instead of using event handlers, you can also poll the status of a thumb stick. You should always cache the reference to your touch controls instead of getting the refence by name every frame to avoid unnecessary overhead.

Thumb stick polling example

```
FastControls.ThumbStick leftThumbStick = null;

void Start() {
    leftThumbStick = myFastControlsInstance.GetThumbStickByName("LeftThumbStick");
}

void Update() {
    if(leftThumbStick != null) {
        if(leftThumbStick.isDown) {
            Debug.Log ("Thumb stick down, direction is " + leftThumbStick.direction
+ " and magnitude " + leftThumbStick.magnitude);
        }
    }
}
```

Sometimes you might want to change the appearance of thumb sticks or buttons at runtime. The following example shows how to hide thumb stick background when the stick is not touched.

Thumb stick background hiding example

```
FastControls.ThumbStick leftThumbStick;
Color thumbStickColor;

void Start() {
    leftThumbStick = myFastControlsInstance.GetThumbStickByName("LeftThumbStick");

    if(leftThumbStick != null) {
        thumbStickColor = leftThumbStick.backgroundTint;

        SetThumbStickBackgroundAlpha(0.0f);
    }
}

void OnEnable() {
    if(leftThumbStick != null) {
        leftThumbStick.OnStickDown += UpdateThumbstickBackgroundVisibility;
        leftThumbStick.OnStickUp += UpdateThumbstickBackgroundVisibility;
    }
}

void OnDisable() {
    FastControls.RemoveAllEventHandlersForObject(this);
}

void UpdateThumbstickBackgroundVisibility(FastControls.ThumbStickEventArgs args) {
    SetThumbStickBackgroundAlpha((args.eventType == FastControls.ThumbStickEventTyp
e.OnStickDown) ? 1.0f : 0.0f);
}

void SetThumbStickBackgroundAlpha(float alpha) {
    if(leftThumbStick != null) {
        thumbStickColor.a = alpha;
        leftThumbStick.backgroundTint = thumbStickColor;
    }
}
```

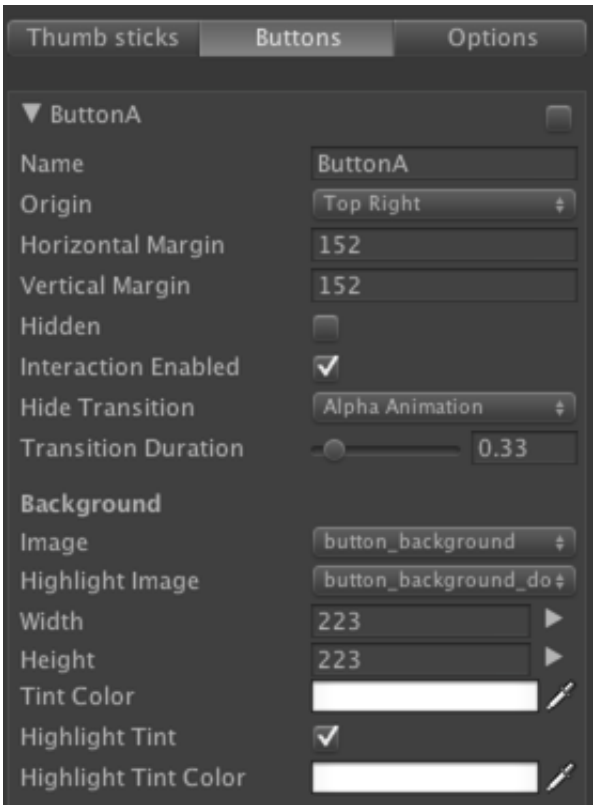
Buttons

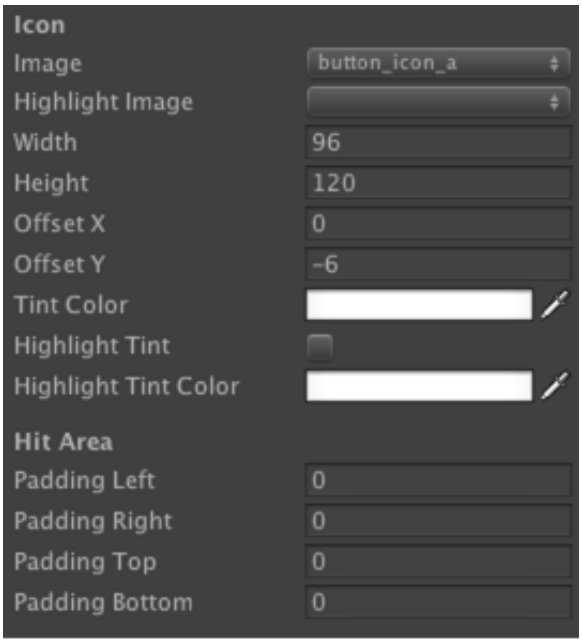
Your FastControls renderer may also contain buttons. A button will require at least one texture asset for the background however a separate icon texture may also be used when needed. Buttons are created almost the same way than thumb sticks and there is no limit how many buttons you can add.

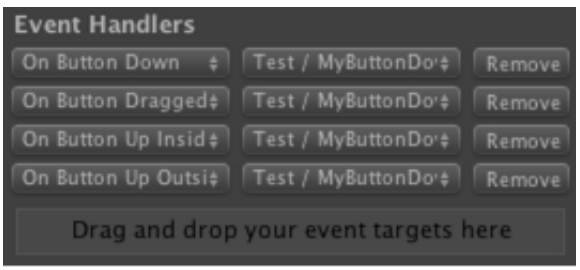
Adding a button

To create a new button, make sure the Buttons tab is selected on your Fast Controls designer.

By clicking the Add button, a new blank button with default properties will be added to the designer. After that you should pick the background texture and set the button's position using the margin. There are many other congfigurable options for buttons too and all the options have tooltips that will tell more information about it.







Listening for button changes

Adding event handlers for a button happens the same way than for thumb sticks. The event

handler methods should just conform to the ButtonEventHandler delegate method instead of ThumbStickEventHandler delegate method.

```
public delegate void ButtonEventHandler(FastControls.ButtonEventArgs args)
```

Just drag and drop the event target on to the designer or subscribe to some or all of the following events at runtime. It is also possible to poll button's isDown property however events give you more information and less performance overhead.

```
public event ButtonEventHandler OnButtonDown
public event ButtonEventHandler OnButtonUpInside
public event ButtonEventHandler OnButtonUpOutside
public event ButtonEventHandler OnButtonDragged
```

Example button event handlers

```
public void ButtonDown(FastControls.ButtonEventArgs args) {
    Debug.Log ("Button down, touch position " + args.touchPosition);
}

public void ButtonUpInside(FastControls.ButtonEventArgs args) {
    Debug.Log ("Button up inside, touch position " + args.touchPosition);
}

public void ButtonUpOutside(FastControls.ButtonEventArgs args) {
    Debug.Log ("Button up outside, touch position " + args.touchPosition);
}

public void ButtonDragged(FastControls.ButtonEventArgs args) {
    Debug.Log ("Button dragged, touch position " + args.touchPosition + ", movement
delta " + args.touchDeltaPosition);
}
```

Runtime button event subscribing example

```
void OnEnable() {
    FastControls.Button buttonA = myFastControlsInstance.GetButtonByName("ButtonA");

    if(buttonA != null) {
        buttonA.OnButtonDown += ButtonDown;
        buttonA.OnButtonUpInside += ButtonUpInside;
        buttonA.OnButtonUpOutside += ButtonUpOutside;
        buttonA.OnButtonDragged += ButtonDragged;
    }
```

```
}

void OnDisable() {
    FastControls.RemoveAllEventHandlersForObject(this);
}
```

Listening for touches

You may also listen for touch and drag events which do not hit any buttons or thumb sticks. These can be used for example to rotate the game world or for picking an object in the 3d world and so on. You may easily implement your own gesture handlers on top of the FastControls touch handler.

The touch event handler methods should conform to the TouchEventHandler delegate method.

```
public delegate void TouchEventHandler(FastControls.TouchEventArgs args)
```

To add a touch event handler you should subscribe to some or all of the following events:

```
public static event TouchEventHandler OnTouchDown
public static event TouchEventHandler OnTouchDragged
public static event TouchEventHandler OnTouchUp
```

Example touch event handlers

```
public void TouchDown(FastControls.TouchEventArgs args) {
    Debug.Log ("Touch down at " + args.touchPosition);
}

public void TouchDragged(FastControls.TouchEventArgs args) {
    Debug.Log ("Touch dragged to " + args.touchPosition + ", movement delta " + arg
s.touchDeltaPosition);
}

public void TouchUp(FastControls.TouchEventArgs args) {
    Debug.Log ("Touch up at " + args.touchPosition);
}
```

Runtime touch event subscribing example

```
void OnEnable() {
    FastControls.OnTouchUp += TouchUp;
```

```
    FastControls.OnTouchDown += TouchDown;
    FastControls.OnTouchDragged += TouchDragged;
}

void OnDisable() {
    FastControls.OnTouchUp -= TouchUp;
    FastControls.OnTouchDown -= TouchDown;
    FastControls.OnTouchDragged -= TouchDragged;
}
```

To detect dragged touches on top of a button or a thumb stick, you can subscribe to the dragged event. The following example will make all buttons and thumb sticks draggable.

Draggable controls example

```
void OnEnable() {
    AddOrRemoveEventHandlers(true);
}

void OnDisable() {
    AddOrRemoveEventHandlers(false);
}

void AddOrRemoveEventHandlers(bool add) {
    foreach(FastControls.Button button in myFastControlsInstance.buttons) {
        if(add) {
            button.OnButtonDragged += DragButton;
        }
        else {
            button.OnButtonDragged -= DragButton;
        }
    }

    foreach(FastControls.ThumbStick thumbStick in myFastControlsInstance.thumbStick
s) {
        if(add) {
            thumbStick.OnStickDragged += DragThumbStick;
        }
        else {
            thumbStick.OnStickDragged -= DragThumbStick;
        }
    }
}

void DragButton (FastControls.ButtonEventArgs args) {
    args.eventSender.Move((int)args.touchDeltaPosition.x, (int)args.touchDeltaPositi
on.y);
}
```

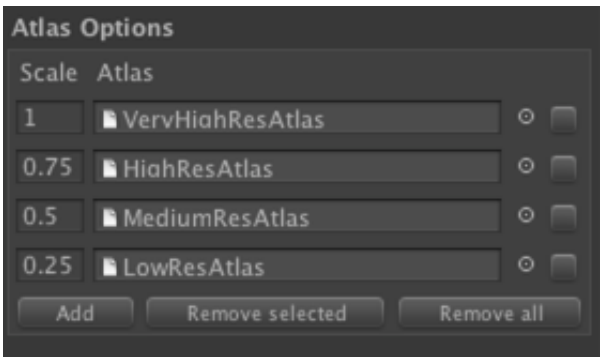
```
void DragThumbStick (FastControls.ThumbStickEventArgs args) {
    args.eventSender.Move((int)args.touchDeltaPosition.x, (int)args.touchDeltaPositi
on.y);
}
```

Scaling

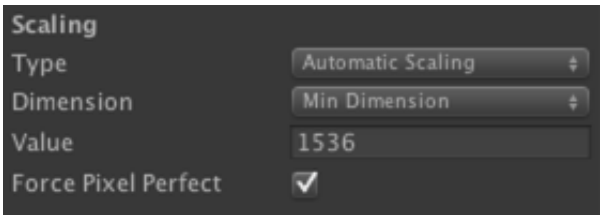
To handle the variety of different screen sizes and resolutions on different devices is not easy. Fast Controls provides a couple of ways to handle it. The easiest way is to use the automatic scaling mode which will make the controls look the same size on all devices related to screen pixel resolution. Automatic scaling also allows you to use different texture atlases for different screen sizes to save memory or to make things pixel perfect. Other option is to use manual scaling and do your own scaling logic, for example based on dpi.

Using the automatic scaling

It is recommended that you design your textures and touch control ui for a large screen size, for example 2048x1536. For the best results or for pixel perfect ui you should create multiple quality texture atlases, e.g low/medium/high/ultrahigh. Fast Controls will then be able to pick the best option for the current scale. If you don't create multiple texture atlases, Fast Controls will use the default atlas and do scaling down or up when rendering the controls. There is no limit how many texture atlases you add.



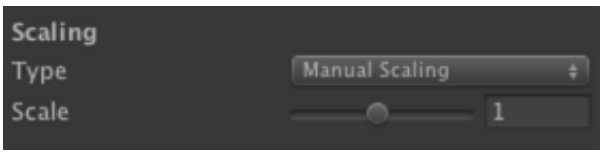
To enable automatic scaling, make sure the Options tab is selected on your designer. Then make sure the scaling type is set to Automatic Scaling. The actual scaling will be calculated based on the current screen size and so called design resolution value that should match to the width or height used when designing the touch control ui (scale 1.0). For most of the cases you should set Dimension to Min Dimension and Value to smallest dimension of your design resolution, for example 1536 if your design resolution was 2048x1536.



If you want to enable pixel perfect rendering, you can check the Force pixel perfect option. This will always round the calculated scale to the nearest atlas option.

Using the manual scaling

Incase you want to use manual scaling, set the scaling type to Manual Scaling. When manual scaling is enabled you can change the scale property of your Fast Controls renderer at runtime like in the example below.



Manual scaling example

```
void Start() {
    // Use your own magic way to set the scale
    if(Mathf.Max(Screen.width, Screen.height) >= 1920) {
        myFastControlsRenderer.scale = 1.0f;
    }
    else {
        myFastControlsRenderer.scale = 0.5f;
    }
}
```

Sharing the atlas

Incase you have multiple Fast Controls renderers it is possible to share the atlas between them. To do this you just need to drag the atlas to the renderer's Default Atlas field. After that you can remove the empty atlas that gets added automatically when adding the actual Fast Controls renderer component. If you are using multiple atlas options, add all the atlas options to your other renderers too.

Input types

Mouse is the default input when the application is running inside the editor. If you wish to use Unity Remote you must change the editor input type to UnityRemote from your Fast Controls

renderer options.

API reference

FastControls

Instance methods

```
public ThumbStick GetThumbStickByName(string name)
public Button GetButtonByName(string name)

public void RemoveEventHandlers()
public void RemoveEventHandlersForObject(object obj)
```

Instance properties

```
public ThumbStick[] thumbSticks
public Button[] buttons
public AtlasOption[] atlasOptions
public FastControlsAtlasBase atlas
public bool hidden
public FastControls.ScalingType scalingType
public FastControls.ScalingDimension automaticScalingDimension
public int automaticScalingBaseValue
public bool automaticScalingForcePixelPerfect
public FastControls.HideTransitionType hideTransitionType
public float hideTransitionDuration
public Shader shader
public float scale
public bool userInteractionEnabled
public FastControls.EditorInputType editorInputType
public bool suppressDebugMessages
```

Static methods (affects all Fast Controls instances)

```
public static void RemoveAllEventHandlers()
public static void RemoveAllEventHandlersForObject(object obj)
```

Static properties

```
public static event TouchEventHandler OnTouchUp
public static event TouchEventHandler OnTouchDragged
public static event TouchEventHandler OnTouchDown
```

Delegate methods

```
public delegate void ThumbStickEventHandler(FastControls.ThumbStickEventArgs args)
public delegate void ButtonEventHandler(FastControls.ButtonEventArgs args)
public delegate void TouchEventHandler(FastControls.TouchEventArgs args)
```

Event types

```
public enum ThumbStickEventType { OnStickDown, OnStickDragged, OnStickUp, OnValuesCh
anged }
public enum ButtonEventType { OnButtonDown, OnButtonUpInside, OnButtonUpOutside, OnB
uttonDragged }
public enum TouchEventType { OnTouchDown, OnTouchDragged, OnTouchUp }
```

FastControls.ThumbStickEventArgs properties

```
public ThumbStickEventType eventType
public ThumbStick eventSender
public int touchId
public Vector2 touchPosition
public Vector2 touchDeltaPosition
```

FastControls.ButtonEventArgs properties

```
public ButtonEventType eventType
public Button eventSender
public int touchId
public Vector2 touchPosition
public Vector2 touchDeltaPosition
```

FastControls.TouchEventArgs properties

```
public TouchEventType eventType
public int touchId
public Vector2 touchPosition
public Vector2 touchDeltaPosition
```

Miscellaneous enums

```
public enum Corner { BottomLeft, BottomRight, TopLeft, TopRight }
public enum ThumbStickMode { Static, TouchArea }
public enum ThumbStickCentering { Instant, LinearAnimation, SmoothStepAnimation, Non
e }
```

```
public enum ThumbStickMaxDistanceType { Inside, Center, Outside }
public enum HideTransitionType { Instant, AlphaAnimation }
public enum ScalingType { ManualScaling, AutomaticScaling }
public enum ScalingDimension { Width, Height, MaxDimension, MinDimension }
public enum EditorInputType { Mouse, UnityRemote }
```

FastControls.ThumbStick

Instance properties

```
public string name
public Corner origin
public int horizontalMargin
public int verticalMargin
public int extendedHitAreaOnLeft
public int extendedHitAreaOnRight
public int extendedHitAreaOnTop
public int extendedHitAreaOnBottom
public int backgroundWidth
public int backgroundHeight
public Color backgroundTint
public bool userInteractionEnabled
public bool requireStickHit
public bool hidden
public FastControls.HideTransitionType hideTransitionType
public float hideTransitionDuration
public int backgroundAtlasSourceIndex
public int hitAreaHorizontalMargin
public int hitAreaVerticalMargin
public int hitAreaWidth
private int mHitAreaHeight
public int thumbStickWidth
public Color thumbStickTint
public bool useCurve
public AnimationCurve curve
public ThumbStickMode mode
private ThumbStickCentering mCenteringType
public float centeringAnimationDuration
public ThumbStickMaxDistanceType stickMaxDistance
public int stickMaxDistanceOffset
public bool highlightTintBackground
public bool limitStickDragDistance
public bool dragBackground
public Color backgroundHighlightTint
public bool highlightTintThumbStick
public Color thumbStickHighlightTint
public int thumbStickAtlasSourceIndex
```

```
public Vector2 direction
public float magnitude
public event ThumbStickEventHandler OnStickDown
public event ThumbStickEventHandler OnStickDragged
public event ThumbStickEventHandler OnStickUp
public event ThumbStickEventHandler OnValuesChanged
public bool isDown
```

Instance methods

```
public void Move(int xDelta, int yDelta)
public void Move(int xDelta, int yDelta, bool applyScale)

public void RemoveEventHandlers()
public void RemoveEventHandlersForObject(object obj)
```

FastControls.Button

Instance properties

```
public string name
public Corner origin
public int horizontalMargin
public int verticalMargin
public int extendedHitAreaOnLeft
public int extendedHitAreaOnRight
public int extendedHitAreaOnTop
public int extendedHitAreaOnBottom
public int backgroundWidth
public int backgroundHeight
public Color backgroundTint
public bool userInteractionEnabled
public bool requireStickHit
public bool showBackgroundOnlyOnTouch
public bool hidden
public FastControls.HideTransitionType hideTransitionType
public float hideTransitionDuration
public int backgroundAtlasSourceIndex
public bool highlightTintBackground
public Color backgroundHighlightTint
public int iconWidth
public int iconHeight
public Color iconTint
public int iconOffsetX
public int iconOffsetY
public bool highlightTintIcon
```

```
public Color iconHighlightTint
public int iconAtlasSourceIndex
public event ButtonEventHandler OnButtonDown
public event ButtonEventHandler OnButtonUpInside
public event ButtonEventHandler OnButtonUpOutside
public event ButtonEventHandler OnButtonDragged
public bool isDown
```

Instance methods

```
public void Move(int xDelta, int yDelta)
public void Move(int xDelta, int yDelta, bool applyScale)

public void RemoveEventHandlers()
public void RemoveEventHandlersForObject(object obj)
```

Support

Incase you are having problems with Fast Controls, don't hesitate to contact

support@pmjo.org