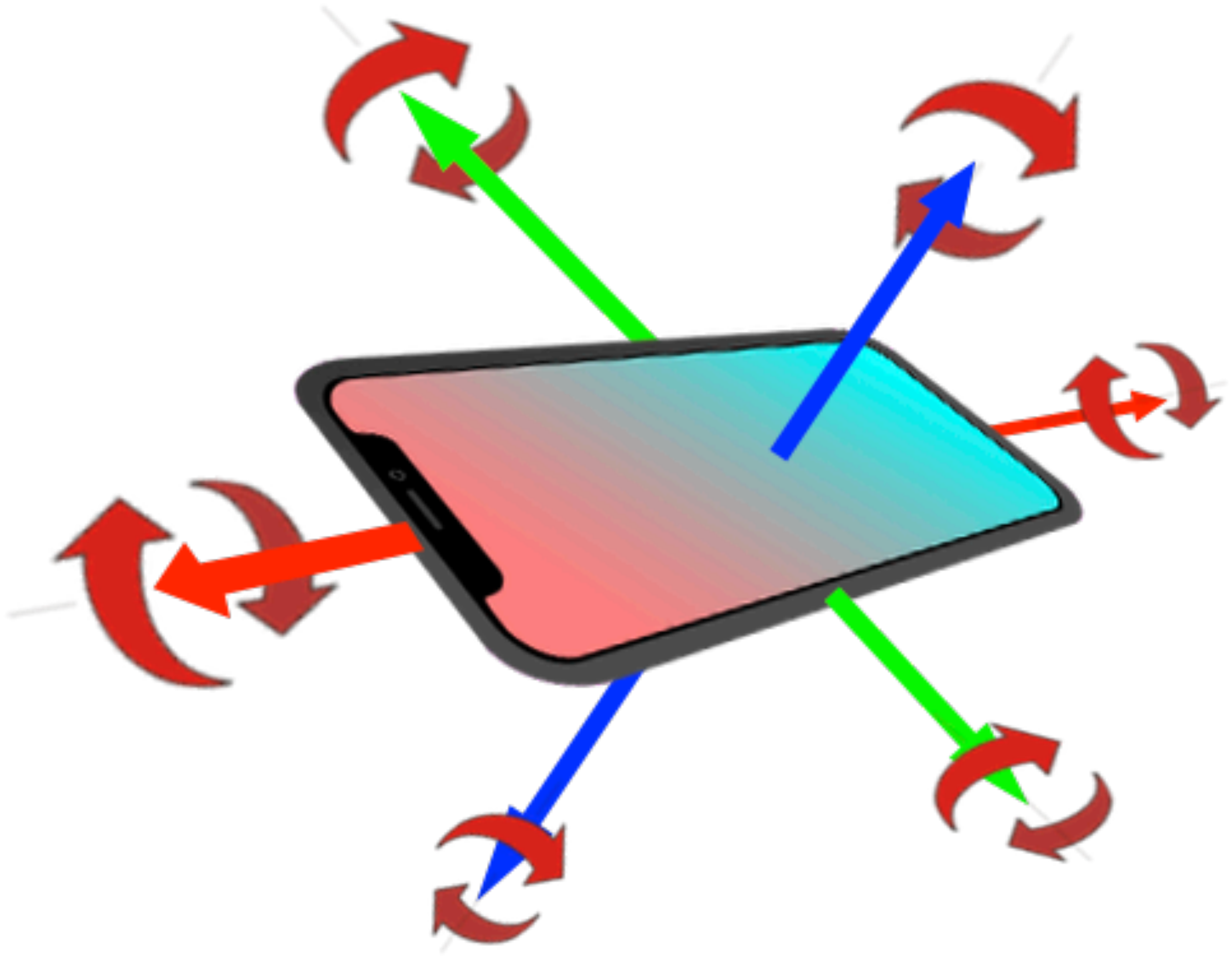
Orientation Sensors Input Controller



[Version 2019.2]

Introduction

Most modern mobile phones and tablets contain various sensors for determining the (absolute) orientation of the device. These are:

- **Accelerometer**. An accelerometer measures *lineair accelerations*, e.g. *changes in velocity* along the x, y and z-axis of the device. Gravity acts like a continuous acceleration upward (also known as Einstein's equivalency principle). Therefor, an accelerometer can also be used to determine which way is up (or down), relative to the devices current orientation. This can be used to calculate a pitch and roll for the device. A rotation around the worlds y-axis (heading) can never be determined with an accelerometer. Accelerometers are very fast and sensitive.
- **Gyroscope.** gyroscopes measure either *changes in orientation* (regular gyro or integrating rate gyro) or *changes in rotational velocity* (rate gyro). Either way, it is used to determine the rotation around all three axis of the device. Gyroscopes usually react very quickly and accurately to rotational changes, but accumulates vast error over time (gyro drift). Also a gyroscope requires knowledge of the starting orientation as a clear point of reference.
- **Magnetometer.** A magnetometer measures the strength and direction of an external magnetic field, relative to the devices current orientation. Because the earth has a rather strong field, the magnetometer can be used as a compass. e.g. to determine an *absolute heading in the NESW plane.* Magnetometer have poor accuracy for fast movements, but pretty much zero drift over time.

Usually the output of two or three of these sensors are combined, because each sensor excels at a different thing. For example, in a digital compass the magnetometer is used to precisely determine the absolute heading, while the device is held steady. But during fast rotations, the gyroscope takes over, using the last known steady value for the heading as a reference starting point. Gyroscopes are also often combined with accelerometers. The gravity vector (from the accelerometer) determines an accurate starting orientation, while the gyroscope determines accurate changes relative to this vector. This way a pitch, roll AND heading of the device can be obtained.
Combining the inputs of all orientation sensors will allow for a quick and accurate determination of the device’s orientation, with a low amount of drift over time.

The main purpose of this asset is to easily acces all the orientation sensors (gyroscope, accelerometer and magnetometer) on a mobile device, and, as an extra bonus, it will allow an user to easily create a simple user interface, with (virtual) joysticks, buttons, touchpads and textures. All scripts and demo’s where tested on several iOS devices. Some customers have confirmed that the scripts also work well on their (non-iOS) devices. I have not tested this myself, though.

The following demo scenes are included:

Demo 1: First Person View
The gyroscope's rotational movements are directly mapped to the (main) camera in the scene, thus creating a First Person View experience when running the game on a handheld device. Move your device to look around in the gameworld or use the touchpad. Use the joystick to move the player forward/backward/left/right.

Demo 2: Turret
A simple Third Person View demo, in which you control a turret/gun. Move/rotate your device to look around in the gameworld, or use the touchpad. Use the red button to shoot at the boxes, use the green button to zoom.

Demo 3: Car

This is a more advanced Third Person View demo, in which you control a simple car. Move/rotate your device to look around in the gameworld. Use the joystick to drive the car, relative to the current rotation of the camera.

Demo 4: Compass
A non-interactive demo, showcasing the possibility to use the magnetometer to build a 3D-compass.

Demo 5: Sword
Use your device as a remote controller, to control a sword on your desktop/laptop screen. Rotate your device and the sword in the gameview of the Unity editor will rotate accordingly. Press the reset button to reset the sword to the center of the screen.

Tip: To make developing using this asset more convenient, use the *latest* "Unity Remote 5" app, . available in the App Store and Google Play, to test your game directly inside the Unity editor.

For further info, you can contact me at: MouseSoftware@GMail.com

Happy Coding!

Maurits.

How to update from a version prior to version 2019.0

Some significant changes were introduced in this version of the asset. Some new features were introduced and most older functionalities have been ported to the new version.
Here are some things you might want to try to allow your older scripts to work with this version:

- The GyroAccel class has been replaced by the all new OrientationSensors class. Replace all references to `GyroAccel` with `OrientationSensors` and most functionality should be restored.
- The VirtualTouchpad, VirtualJoystick and VirtualButton classes have been replaced by a single new TouchInterface Class. Search all your scripts and find all references to `VirtualTouchpad`, `VirtualJoystick` and `VirtualButton`. Replace these with `TouchInterface`, to restore most of the functionality.
- The current version no longer supports the option to use SendMessage to automatically and constantly send messages to gameobjects. This rather outdated functionality turned out to be rather slow. Instead, directly use the API in your scripts.

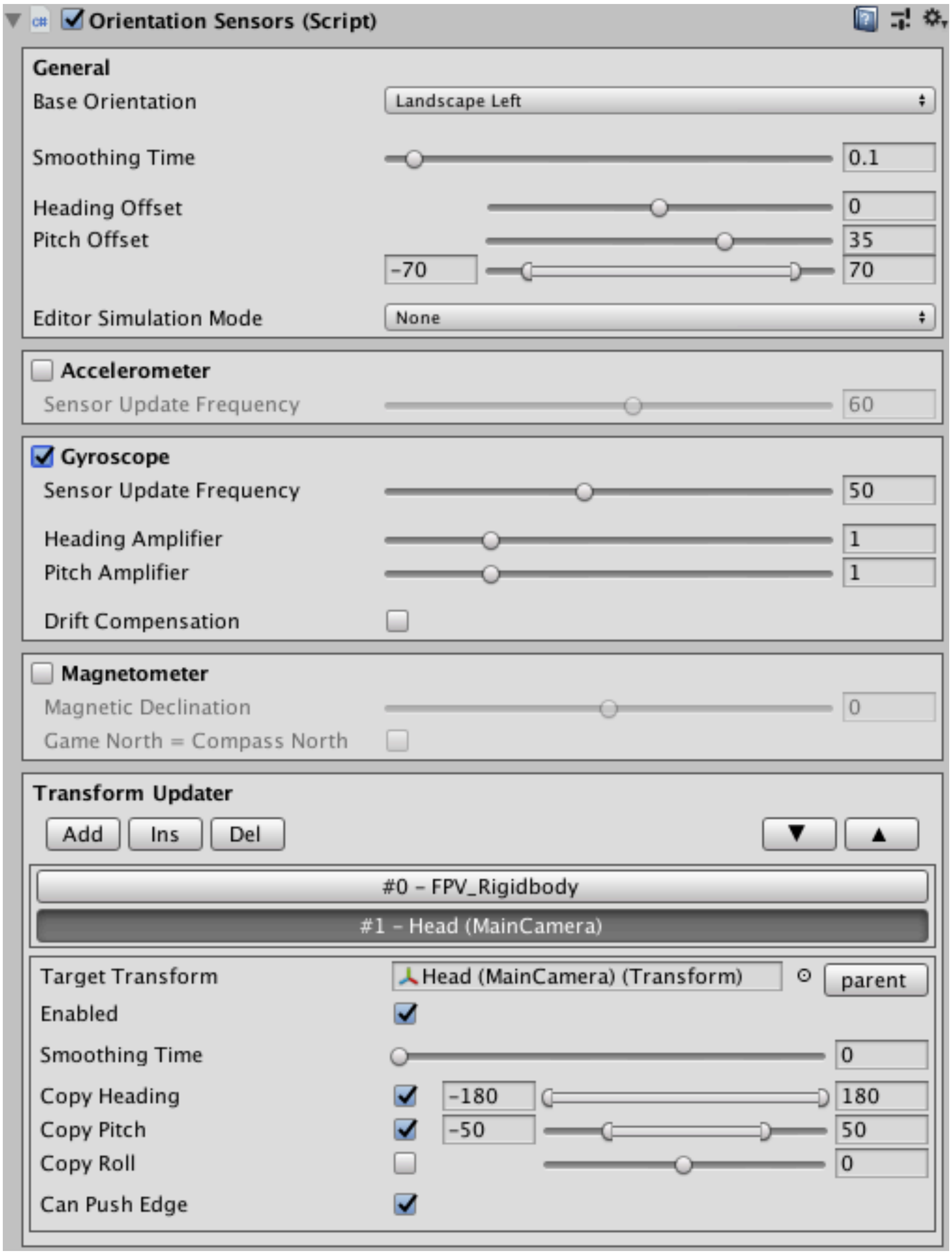
Quickstart

Put the “MSP_Input” prefab in the scene Hierarchy. This prefab contains Several scripts. Each script can be easily configured in Unity’s inspector.

- *OrientationSensors.cs*: The main script of this asset, which processes all gyroscope and accelerometer sensor output.
- *TouchInterface.cs:* This script allows the creation of Joysticks, Touchpads, Buttons and Textures, while working in the editor. During gameplay, it processes all user input from these UI elements.

Important: Make sure there is only *one* instance of each of these scripts active in the scene at all time.

OrientationSensors.cs



Orientation Sensors (Script)

General

Base Orientation	Landscape Left
Smoothing Time	0.1
Heading Offset	0
Pitch Offset	35
	-70 – 70
Editor Simulation Mode	None

☐ Accelerometer

Sensor Update Frequency	60

✓ Gyroscope

Sensor Update Frequency	50
Heading Amplifier	1
Pitch Amplifier	1
Drift Compensation	☐

☐ Magnetometer

Magnetic Declination	0
Game North = Compass North	☐

Transform Updater

Add Ins Del ▼ ▲

#0 – FPV_Rigidbody

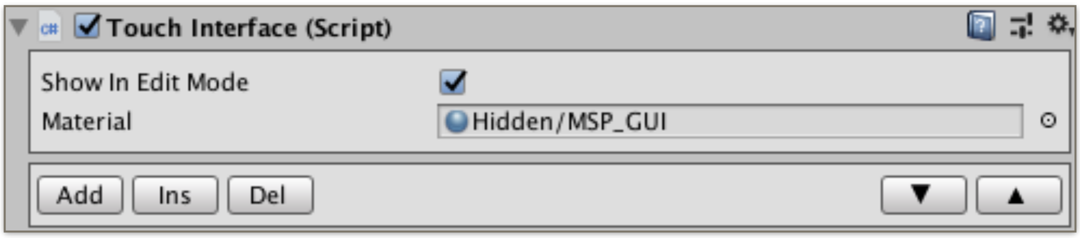
#1 – Head (MainCamera)

Target Transform	Head (MainCamera) (Transform)		parent
Enabled	✓		
Smoothing Time			0
Copy Heading	✓	-180	180
Copy Pitch	✓	-50	50
Copy Roll	☐		0
Can Push Edge	✓		

General	
Base Orientation	The base orientation to be used during gameplay. Most common modes are Portrait and Landscape Left/Right, where the device is being held in front of the player, like a viewport to the gameworld. Pitch and roll are calculated relative to the gravity vector, the heading is calculated relative to the device's forward vector. However, when looking straight up or down, the value for heading is not well defined (which is a limitation of using Euler angles, not a limitation of the asset). If the device is being used on a flat surface ('face up') this will result in unwanted behavior. In these cases, you'd better select one of the alternative, e.g Face Up, orientations. The heading will be calculated with respect to the devices up-vector. The compass and the sword demo both use these modes. Note: Changing this value will also change the selected orientation mode in Player Settings —> Resolution and presenation —> Default Orientation
Smoothing Time	The (general) smoothing time to be used.
HeadingOffset	The heading offset to be used (North = 0, 90 = east, 180 = south, 270 = west, etc.). Please note that this value can/will be changed during runtime, either by yourself or internally by the script, e.g. when the HeadingAmplifier is being used.
PitchOffset	The pitch offset to be used (straight up = 90, level = 0, straight down = -90). Here you can also select the minimum/maximum values of the pitchOffset. For playability issues, don't use values near 90 degrees. Please note that the value for the PitchOffset can/will be changed during runtime, either by yourself or internally by the script, e.g. when the PitchAmplifier is being used.
Editor Simulation Mode	Select the mouse or cursor keys to simulate gyro movements in the editor.
Accelerometer	Select this to enable the accelerometer. The gyroscope will be automatically deactivated
Sensor Update Frequency	The update frequency of the accelerometer. Higher values might result in higher accuracy, but will also drain the battery more quickly. Typical values are 15Hz, 30Hz, 60Hz or 100Hz.
Gyroscope	Select this if you want to use the gyroscope. The accelerometer will be automatically deactivated. (Well, not entirely: Internally, the accelerometer is still required to determine the gravity vector).
Sensor Update Frequency	The update frequency of the gyroscope. Higher values might result in higher accuracy, but will also drain the battery more quickly. In some rare cases, high values might result in situations where data could be "leaking through", as the updates by the iOS Core Motion Framework appear to be happening faster than the execution of the rest of the code. For typical usage, a value of ±50Hz should be fine.

Gyro Heading Amplifier *Gyro Pitch Amplifier*	The heading and pitch multipliers for the gyroscope. Setting these values >1 will effectively speed up the rotation of your character in the gameworld. Using values < 1 will slow it down.
Drift Compensation [EXPERIMENTAL]	If selected, the script will attempt to compensate for gyro drift, by combining the input from all three orientation sensors.
Magnetometer	Select this if you want to use the magnetometer. It will be automatically enabled when selecting the gyroscope's drift compensation, as it plays an essential part.
Magnetic Declination	The angle on the horizontal plane between Magnetic North and Geographic / True North is called Magnetic Declination. This angle varies depending on position on the Earth's surface and changes over time. Go to http://www.magnetic-declination.com to find the magnetic declination at your position.
Game North = Compass North	Automatically sets the Game North to the Compass North. The gyroscopes's heading- and pitch amplifiers will be reset to 1.
Transform Updater	
Add, Ins, Del	Add, insert or delete Transforms that will have their orientations automatically updated during gameplay
▼*,* ▲	Use the sorting buttons to customize the order in which the transform are being updated. (Usually, the main camera will come last).
Target Transform	The Transform that will have it's orientation updated during gameplay. Press the 'parent' button to select the parent transform.
Enabled	The rotational update for this transform is enabled/disabled.
Smoothing Time	An (additional) smoothing time can be set; the smoothing time in the general settings will always be applied.
Copy Heading *Copy Pitch* *Copy Roll*	Select if the heading and/or the pitch and/or the roll of the transform should be updated. Restrictions can be set for the minimum and maximum values of heading, pitch and roll, for each transform. If you choose to partially update the rotation (e.g. only the heading, pitch and/or roll will be updated), you can choose a default value for the remaining axis.
Can Push Edge	When a heading or pitch moves beyond it's boundaries, you can choose to 'push the edge' along with it. This will effectively recalculate all offset variables, and might thereby influence all other game objects being controlled by this asset.

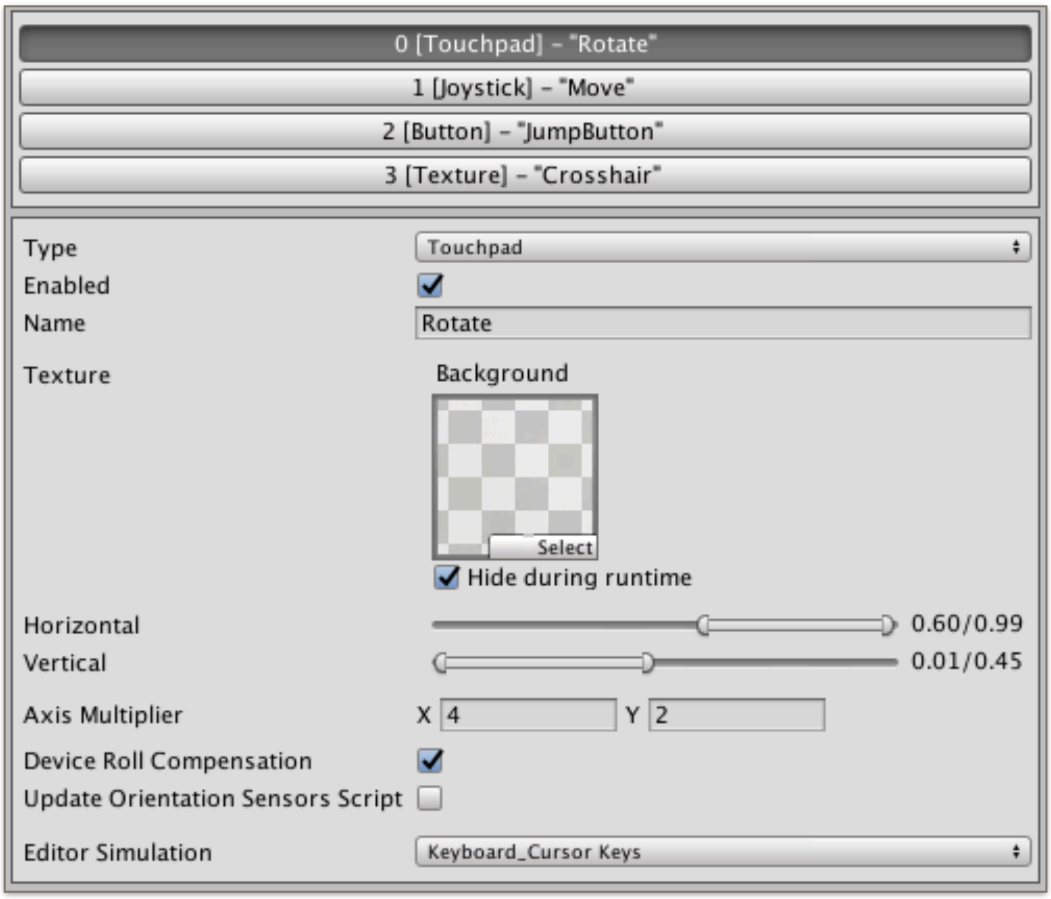
TouchInterface.cs



Show In Edit Mode ✓
Material Hidden/MSP_GUI

Show In Edit Mode	Quickly show or hide all UI elements in edit mode
Material	The material to be used for all UI elements. If none selected, the default MSP_GUI shader will be used (a very basic shader, supporting transparent textures)
Add, Ins, Del	Add, insert or delete UI elements.
▼, ▲	Use the sorting buttons to customize the order in which the UI elements are being drawn on the screen.

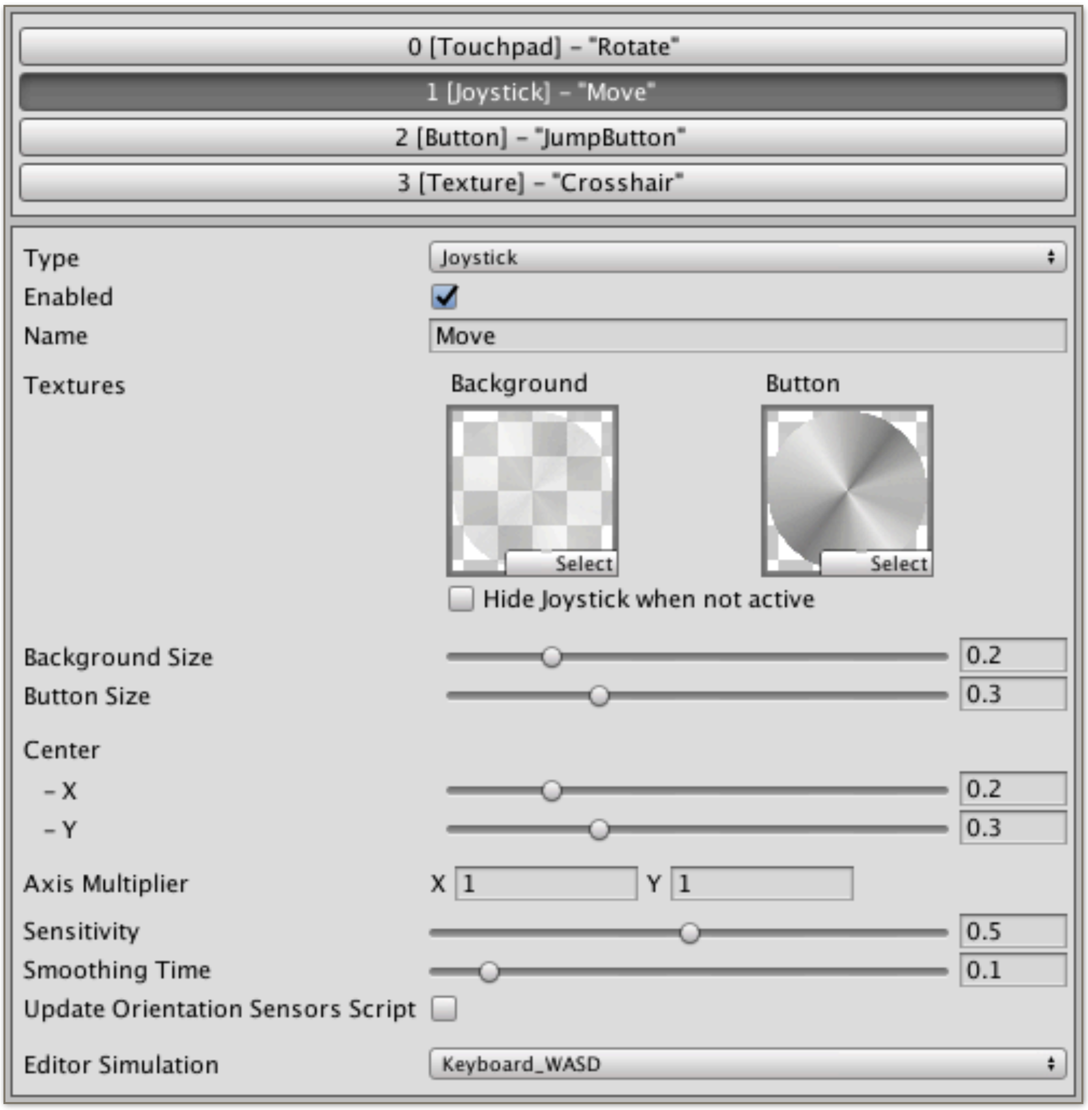
TouchInterface.cs —> Touchpad


0 [Touchpad] - "Rotate"
1 [Joystick] - "Move"
2 [Button] - "JumpButton"
3 [Texture] - "Crosshair"

Type	Touchpad
Enabled	✓
Name	Rotate
Texture	Background; ✓ Hide during runtime
Horizontal	0.60/0.99
Vertical	0.01/0.45
Axis Multiplier	X 4 Y 2
Device Roll Compensation	✓
Update Orientation Sensors Script	☐
Editor Simulation	Keyboard_Cursor Keys

Type	Four types of UI elements are supported: Touchpad, Joystick, Button and Texture (Here: Touchpad)
Enabled	Enable or disable this UI element.
Name	The name of this touchpad. Make sure you use the exact same name when referring to this UI element when using it elsewhere (e.g. when using the `MSP_Input.TouchInterface.GetID()` command)
Texture	The texture to be used for this touchpads background. You can choose to hide this texture during gameplay.
Horizontal / Vertical	Use the sliders to control the position and size of the touchpad on the screen. Given values are in relative screen coordinates: 0 is left / bottom of the screen, while 1 is right / top of the screen.
Axis Multiplier	By default, the touchpad returns a Vector2 with values between -1 and 1. These values can be multiplied with an axisMultiplier. Tip: if you want to invert the touchpad's movement, use negative values for the axisMultiplier
Device Roll Compensation	Should the output be compensated for the roll of your device. e.g.: when turning your device like a steering wheel, the input direction of the touchpad will follow accordingly.
Update Orientation Sensors Script	Use this touchpad to control the heading- and pitch-offset values of the OrientationSensors script.
Editor Simulation	Select one of the key-configurations to simulate touch and/or doubletap.

TouchInterface.cs —> Joystick

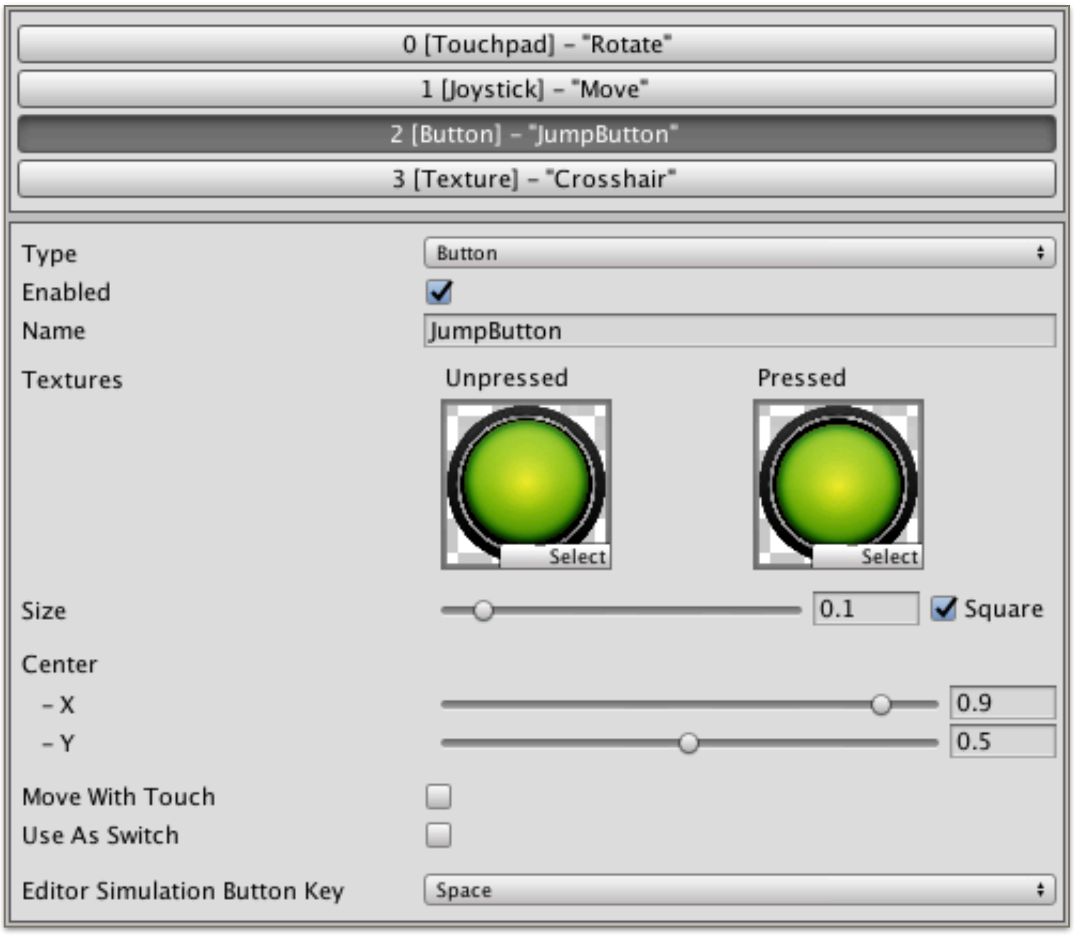

0 [Touchpad] - "Rotate"
1 [Joystick] - "Move"
2 [Button] - "JumpButton"
3 [Texture] - "Crosshair"

Type	Joystick
Enabled	✓
Name	Move
Textures	Background, Button
	☐ Hide Joystick when not active
Background Size	0.2
Button Size	0.3
Center	
- X	0.2
- Y	0.3
Axis Multiplier	X 1 Y 1
Sensitivity	0.5
Smoothing Time	0.1
Update Orientation Sensors Script	☐
Editor Simulation	Keyboard_WASD

Type	Four types of UI elements are supported: Touchpad, Joystick, Button and Texture (Here: Joystick)
Enabled	Enable or disable this UI element.
Name	The name of this UI element. Make sure you use the exact same name when referring to this UI element when using it elsewhere (e.g. when using the `MSP_Input.TouchInterface.GetID()` command)
Background Texture *Button Texture*	The texture to be used for the joystick's background and button. You can choose to hide these texture during gameplay, when the joystick is inactive.
Background Size	The size of the background texture, relative to the screens width (1 = full screen width).
Button Size	The size of the button texture, relative to the size of the background (1 = same size as the background texture).
Center X *Center Y*	The center of this UI element. Given values are in relative screen coordinates: 0 is left / bottom of the screen, while 1 is right / top of the screen.

Axis Multiplier	By default, the joystick returns a Vector2 with values between -1 and 1. These values can be multiplied with an axisMultiplier. Tip: if you want to invert the joystick movement, use negative values for the axisMultiplier
Sensitivity	The sensitivity of the joystick; how much should the joystick respond when you’re moving it just a little to the outside? Smaller values will make the joystick less sensitive while the button is near the middle.
Smoothing Time	The (maximum) time in which this UI element moves towards it target position.
Update Orientation Sensors Script	Use this UI element to control the heading- and pitch-offset values of the OrientationSensors script.
Editor Simulation	Select one of the key-configurations to simulate touch and/or doubletap.

TouchInterface.cs —> Button

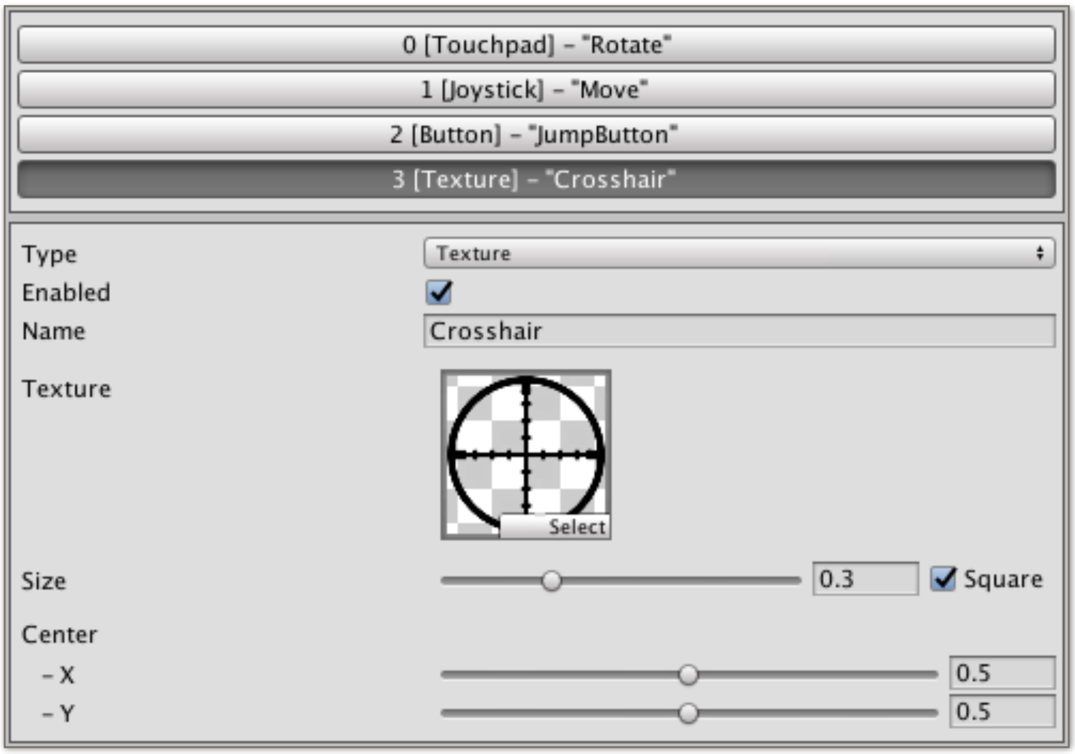

0 [Touchpad] - "Rotate"
1 [Joystick] - "Move"
2 [Button] - "JumpButton"
3 [Texture] - "Crosshair"

Type: Button
Enabled: ✓
Name: JumpButton
Textures: Unpressed, Pressed
Size: 0.1 ✓ Square
Center
- X: 0.9
- Y: 0.5

Move With Touch: ☐
Use As Switch: ☐
Editor Simulation Button Key: Space

Type	Four types of UI elements are supported: Touchpad, Joystick, Button and Texture (Here: Button)
Enabled	Enable or disable this UI element.
Name	The name of this UI element. Make sure you use the exact same name when referring to this UI element when using it elsewhere (e.g. when using the `MSP_Input.TouchInterface.GetID()` command)
Unpressed Texture *Pressed Texture*	The texture to be used for the button's pressed and unpressed state.
Size	The size of this UI element, relative to the screen's size (1 = full screen width). When 'square' is deselected, the size must be given in separate x- and y-values (1 = full screen width / height).
Center X *Center Y*	The center of this UI element. Given values are in relative screen coordinates: 0 is left / bottom of the screen, while 1 is right / top of the screen.
Move With Touch	Once pressed, should this UI element move along with the touch input? A Smoothing Time can be given, which is the (maximum) time in which this UI element moves towards it target position.
Use As Switch	If selected, the button will behave like a switch.
Editor Simulation	Select a key to simulate the button.

TouchInterface.cs —> Texture



0 [Touchpad] - "Rotate"
1 [Joystick] - "Move"
2 [Button] - "JumpButton"
3 [Texture] - "Crosshair"

Type	Texture
Enabled	✓
Name	Crosshair
Texture	Select
Size	0.3 ✓ Square
Center	
- X	0.5
- Y	0.5

Type	Four types of UI elements are supported: Touchpad, Joystick, Button and Texture (Here: Texture)
Enabled	Enable or disable this UI element.
Name	The name of this UI element.
Texture	The texture to be used for this UI element.
Size	The size of this UI element, relative to the screen's size (1 = full screen width). When 'square' is deselected, the size must be given in separate x- and y-values (1 = full screen width / height).
Center X *Center Y*	The center of this UI element. Given values are in relative screen coordinates: 0 is left / bottom of the screen, while 1 is right / top of the screen.

Advanced (additional coding with API)

Each script, once active in the scene, will continuously update it’s parameters and variables. Each script contains various public static functions, which can be used to read/write these parameters and variables. All available commands of this API are summarized at the end of this document. First some short examples (c#):

Example 1 - Apply the gyro’s rotation to a transform

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour
{
	public Transform someTransform;

	void Update ()
	{
		someTransform.rotation = MSP_Input.OrientationSensors.GetRotation();
	}
}
```

Example 2 - Use the mouse to add an extra heading/pitch to the gyroscope’s offset

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour
{
	void Update ()
	{
		float mouseX = Input.GetAxis("Mouse X”);
		float mouseY = Input.GetAxis("Mouse Y”)

		MSP_Input.OrientationSensors.AddFloatToHeadingOffset(mouseX);
		MSP_Input.OrientationSensors.AddFloatToPitchOffset(mouseY);
	}
}
```

Example 3 - Start a function when a TouchInterface with name “button1” is pressed

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour
{
      private int buttonID;

      void Start ()
      {
            // cache the button’s unique ID-number
            buttonID = MSP_Input.TouchInterface.GetID(“button1”)
      }

      void Update ()
      {
            if (MSP_Input.TouchInterface.GetButtonDown(buttonID))
            {
                  DoSomething();
            }
      }

      void DoSomething()
      {
            // do stuff
      }
}
```

Example 4 - Read the axis of a TouchInterface with name “moveJoystick” and move a gameobject along it’s (x,z)-axis

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour
{

      public Transform transformToMove;
      private int joystickID;

      void Start ()
      {
            // cache the joystick’s unique ID-number
            joystickID = MSP_Input.TouchInterface.GetID(“moveJoystick”)
      }

      void Update ()
      {
            Vector2 axis = MSP_Input.TouchInterface.GetAxis(joystickID);
            Vector3 move = new Vector3(axis.x,0f,axis.y);
            transformToMove.Translate(move * Time.deltaTime, Space.Self);
    }
}
```

Note: You can also put “using MSP_Input;” at the top of a script. Instead of writing something like “MSP_Input.OrientationSensors.GetRotation()”, You can now simply use “OrientationSensors.GetRotation()”.

PUBLIC STATIC FUNCTIONS FOR *MSP_Input.OrientationSensors*:

bool MSP_Input.OrientationSensors.IsGyroscopeEnabled()
returns true if the gyroscope is enabled.

bool MSP_Input.OrientationSensors.IsAccelerometerEnabled()
returns true if the gyroscope is enabled.

bool MSP_Input.OrientationSensors.IsMagnetometerEnabled()
returns true if the gyroscope is enabled.

bool MSP_Input.OrientationSensors.IsGyroDriftCompensationEnabled()
returns true if the drift compensation for the gyro is enabled. [experimental]

void MSP_Input.OrientationSensors.GyroscopeEnabled(bool enabled)
`GyroscopeEnabled(true)` will enable the gyroscope, and disable the accelerometer. `GyroscopeEnabled(false)` will disable the gyroscope, and enable the accelerometer.

void MSP_Input.OrientationSensors.AccelerometerEnabled(bool enabled)
`AccelerometerEnabled(true)` will enable the accelerometer, and disable the gyroscope. `AccelerometerEnabled(false)` will disable the accelerometer, and enable the gyroscope.

void MSP_Input.OrientationSensors.MagnetometerEnabled(bool enabled)
`MagnetometerEnabled(true)` will enable the magnetometer. `MagnetometerEnabled(false)` will disable the magnetometer.

void MSP_Input.OrientationSensors.SetBaseOrientation(BaseOrientation newBaseOrientation)
Set the Base Orientation of the device to one of the following: MSP_Input.OrientationSensors.BaseOrientation.Portrait, MSP_Input.OrientationSensors.BaseOrientation.LandscapeLeft, MSP_Input.OrientationSensors.BaseOrientation.LandscapeRight, MSP_Input.OrientationSensors.BaseOrientation.Portrait_FaceUp, MSP_Input.OrientationSensors.BaseOrientation.LandscapeLeft_FaceUp, MSP_Input.OrientationSensors.BaseOrientation.LandscapeRight_FaceUp

void MSP_Input.OrientationSensors.SetGyroDriftCompensation(bool enabled)
`SetGyroDriftCompensation(true)` will enable drift compensation for the gyroscope (experimental). `SetGyroDriftCompensation(false)` will disable drift compensation for the gyroscope.

Quaternion MSP_Input.OrientationSensors.GetRotation()
returns the current rotation (Quaternion)

float MSP_Input.OrientationSensors.GetHeading()
returns the current heading, clamped between -180 and 180 degrees (Forward = 0, Right = 90, Back = (-)180 and Left = -90)

float MSP_Input.OrientationSensors.GetHeadingUnclamped()
returns the current unclamped heading

void MSP_Input.OrientationSensors.SetHeading(float newHeading)
set/force the current heading to the value of *newHeading.*

float MSP_Input.OrientationSensors.GetHeadingOffset()
Returns the current value of the headingOffset

void MSP_Input.OrientationSensors.SetHeadingOffset(float newHeadingOffset)
Sets the headingOffset to it's new value of *newHeadingOffset*

void MSP_Input.OrientationSensors.AddFloatToHeadingOffset(float extraHeadingOffset)
Adds an extra value of *extraHeadingOffset* to the current heading offset

float MSP_Input.OrientationSensors.GetPitch()
returns the current pitch, always between -90 (up) and 90 (down) degrees.

void MSP_Input.OrientationSensors.SetPitch(float newPitch)
set/force the current pitch to the value of *newPitch.* Note: this might not always give the expected result; during the calculation of the pitch, the pitchOffset boundaries put a final restriction on the allowed values for the pitch.

float MSP_Input.OrientationSensors.GetPitchOffset()
returns the current value of the pitchOffset

void MSP_Input.OrientationSensors.SetPitchOffset(float newPitchOffset)
Sets the pitchOffset to it's new value of *newPitchOffset*

void MSP_Input.OrientationSensors.AddFloatToPitchOffset(float extraPitchOffset)
Adds an extra value of *extraPitchOffset* to the current pitch offset

void MSP_Input.OrientationSensors.SetPitchOffsetMinumumMaximum(float newPitchOffsetMinimum, float newPitchOffsetMaximum)
Set the minimum and maximum value of the pitchOffset. For playability issues, don’t use values near 90 degrees.

float MSP_Input.OrientationSensors.GetRoll()
returns the current roll, clamped between -180 and 180 degrees (clockwise/counterclockwise)

float MSP_Input.OrientationSensors.GetRollUnclamped()
returns the current unclamped roll

void MSP_Input.OrientationSensors.GetHeadingPitchRoll(out float h, out float p, out float r)
gets all the values for heading, pitch and roll

void MSP_Input.OrientationSensors.GetHeadingPitchRollUnclamped(out float h, out float p, out float r)
gets all the unclamped values for heading, pitch and roll

float MSP_Input.OrientationSensors.GetCompassHeading360()
returns the current value of the compass heading, clamped between 0 and 360 degrees.

float MSP_Input.OrientationSensors.GetCompassHeading180()
returns the current value of the compass heading, clamped between -180 and 180 degrees.

void MSP_Input.OrientationSensors.SetGameNorthToCompassNorth(float lerpFactor)
Lerps the Game North towards the Compass North

void MSP_Input.OrientationSensors.SetGameNorthToCompassNorth(bool setAutomatically)
Sets the Game North directly to the Compass North. If setAutomatically == true, this will be done during every Update cycle.

void MSP_Input.OrientationSensors.SetGameNorthToCompassNorth(float lerpFactor, bool setAutomatically)
Lerps the Game North towards the Compass North. If setAutomatically == true, this will be done during every Update cycle.

float MSP_Input.OrientationSensors.GetMagneticDeclinationAtCurrentLocation()
returns the magnetic declination at the current location, derived from the difference between the values of `Input.compass.trueHeading` and `Input.compass.magneticHeading`

float MSP_Input.OrientationSensors.GetSmoothingTime()
Returns the current smoothing time

void MSP_Input.OrientationSensors.SetSmoothingTime(float smoothTime)
Sets the smoothing time. A (change in rotation) will be applied smoothly during this time.

float MSP_Input.OrientationSensors.GetGyroHeadingAmplifier()
returns the current value of the gyroscope's heading amplifier.

void MSP_Input.OrientationSensors.SetGyroHeadingAmplifier(float newValue)
Set the gyroscope's heading amplifier to a value of *newValue.* *newValue* < 1 —> decreases a change in the gyroscopes heading *newValue* = 1 —> keeps the gyroscope's change of heading unaltered *newValue* > 1 —> increases a change in the gyroscopes heading

float MSP_Input.OrientationSensors.GetGyroPitchAmplifier()
returns the current value of the gyroscope's pitch amplifier.

void MSP_Input.OrientationSensors.SetGyroPitchAmplifier(float newValue)
Set the gyroscope's pitch amplifier to a value of *newValue.* *newValue* < 1 —> decreases a change in the gyroscopes pitch *newValue* = 1 —> keeps the gyroscope's change of pitch unaltered *newValue* > 1 —> increases a change in the gyroscopes pitch

float MSP_Input.OrientationSensors.ConvertAngle180To360(float angle180)
Takes an input angle with a range of -180 degrees to 180 degrees, and converts this to an angle with a range from 0 degrees to 360 degrees.

float MSP_Input.OrientationSensors.ConvertAngle360To180(float angle360)
Takes an input angle with a range of 0 degrees to 360 degrees, and converts this to an angle with a range from -180 degrees to 180 degrees.

Quaternion MSP_Input.OrientationSensors.GetQuaternionFromHeadingPitchRoll(float inputHeading, float inputPitch, float inputRoll)
returns a Quaternion, by first applying an *inputHeading*, then Applying an (local) *inputPitch* and then applying an (local) *inputRoll.*

int MSP_Input.OrientationSensors.GetTransformUpdaterID(Transform sourceTransform)
Get a unique ID, linked to sourceTransform.

bool MSP_Input.OrientationSensors.EnableTransformUpdate(Transform targetTransform) bool MSP_Input.OrientationSensors.EnableTransformUpdate(int id)
Disables automatic rotation updates for the targetTransform, or the Transform with unique id *id.* Returns true if successful, or false otherwise.

bool MSP_Input.OrientationSensors.DisableTransformUpdate(Transform targetTransform) bool MSP_Input.OrientationSensors.DisableTransformUpdate(int id)
Disables automatic rotation updates for the targetTransform, or the Transform with unique id *id.* Returns true if successful, or false otherwise.

PUBLIC STATIC FUNCTIONS FOR *MSP_Input.TouchInterface*:

int MSP_Input.TouchInterface.GetID(string name)
Returns the unique identifier, associated with the UI element with name *name*. It’s recommended to use this identifier during gameplay, as it ensures less work for the garbage collector.

bool MSP_Input.TouchInterface.Enable(int id) bool MSP_Input.TouchInterface.Enable(string name)
Enables the UI element with identifier *id* (or name *name*). Returns true if successful.

bool MSP_Input.TouchInterface.Disable(int id) bool MSP_Input.TouchInterface.Disable(string name)
Disables the UI element with identifier *id* (or name *name*). Returns true if successful. Disabled UI elements are no longer visible on screen but can easily be (re)activated by using the Enable command.

bool MSP_Input.TouchInterface.GetButton(int id) bool MSP_Input.TouchInterface.GetButton(string name)
Returns *true* if the UI element with identifier *id* (or name *name*) is being pressed.

bool MSP_Input.TouchInterface.GetButtonDown(int id) bool MSP_Input.TouchInterface.GetButtonDown(string name)
Returns *true* if the UI element with identifier id (or name *name*) has just been pressed. Once the UI element is down, this function will return *false*

bool MSP_Input.TouchInterface.GetButtonUp(int id) bool MSP_Input.TouchInterface.GetButtonUp(string name)
Returns *true* if the UI element with identifier id (name *name*) has been released.

bool MSP_Input.TouchInterface.GetDoubleTap(int id) bool MSP_Input.TouchInterface.GetDoubleTap(string name)
For joysticks and touchpads: returns true if the UI element with the identifier id (or name *name*) has been double tapped during the last update-cycle. For all other Ui elements: result will always return false.

bool MSP_Input.TouchInterface.GetDoubleTapHold(int id) bool MSP_Input.TouchInterface.GetDoubleTapHold(string name)
For joysticks and touchpads: returns true if the UI element with the identifier id (or name *name*) has been double tapped during the last update-cycle. The returned value remains true, until the TouchInterface has been released. For all other Ui elements: result will always return false.

Vector2 MSP_Input.TouchInterface.GetAxis(int id)
Vector2 MSP_Input.TouchInterface.GetAxis(string name)

Returns a Vector2 with the current value of the axis of the UI element with the identifier *id* (or name *name*).
For joysticks: The axis is defined as a Vector2 with x,y-values between -1 and 1, indicating the direction and magnitude of the joystick's movement relative to it's center. These values are then multiplied with the according axisMultiplier's
For touchpads: The axis is defined as a Vector2 with x,y-values between -1 and 1, indicating the direction and magnitude of the touchpad movement relative to where the player has touched the touchpad in the last update. These values are then multiplied with an axisMultiplier.
For all other UI elements: result will always return Vector2.zero;

void MSP_Input.TouchInterface.GetAngleAndMagnitude(int id, out float angle, out float magnitude)
void MSP_Input.TouchInterface.GetAngleAndMagnitude(string name, out float angle, out float magnitude)

Get the current status of the UI element with identifier *id* (or name *name*):
For Joysticks: The joysticks angle (-180 < angle < 180) is returned with the *angle* variable.
Its magnitude (0 < magnitude < 1) is returned with the *magnitude* variable.
For all other UI elements: result will always return angle = 0; magnitude = 0

Vector2 MSP_Input.TouchInterface.GetAxisMultiplier(int id)
Vector2 MSP_Input.TouchInterface.GetAxisMultiplier(string name)

For joysticks and touchpads: returns the current values for the axis multipliers.
For all other UI elements: result will always return zero.

bool MSP_Input.TouchInterface.SetAxisMultiplier(int id, Vector2 axisMultiplier)
bool MSP_Input.TouchInterface.SetAxisMultiplier(string name, Vector2 axisMultiplier)

For joysticks and touchpads: sets new values for the axis multipliers to be used. Returns true if successful.
For all other UI elements: result will always return false.

bool MSP_Input.TouchInterface.SetSensitivity(int id, float sensitivity)
bool MSP_Input.TouchInterface.SetSensitivity(string name, float sensitivity)

For joysticks: sets a new values for the joysticks sensitivity. Returns true if successful.
For all other UI elements: result will always return false.

Changelog

v2019.2

- Quick fix for a bug in Unity (Case 1177775, where the gyroscope might not work correctly in iOS 13 and iPadOS 13. Previous versions are unaffected). This fix only works when your app is directly executed on a iPhone or iPad, through an Xcode build. Working with the current version of the Unity Remote 5 App (v2.0) inside Unity's editor does not work. Unity is aware of the situation and is working on a solution.
- The gyroscope's Sensor Update Frequency can now be set in the Inspector. The default value is 50Hz (was 100Hz)
- Also added the option in the inspector to set the Accelerometer's Sensor Update Frequency, with a default value of 60Hz.

v2019.1

- IMPORTANT: as always, first make a backup of your project, before updating.
- Fixed an error when importing the asset
- Fixed some bugs in Portrait_FaceUp and Landscape_FaceUp Base Orientations.
- Portrait_FaceUp and Landscape_FaceUp Base Orientations work best with a pitchOffset = 0, so this is now the default value.
- Changing values during runtime sometimes could result in unwanted behaviour. This has now been resolved, by disabling parts of the inspector while in runtime mode.

v2019.0

- IMPORTANT: This is a huge update and this version is NOT backwards compatible with previous versions. ALWAYS MAKE A BACKUP BEFORE UPDATING.
  Most older projects can be updated with this version, but it will require some changes in your scripts. See the section "How to update from versions prior to v2019" in the manual.
- Name of the asset was changed form "First Person View Gyroscope/Accelerometer Input Controller" to "Orientation Sensors Input Controller" to better reflect the general usage of the product.
- A rewrite of most of the scripts. in order to keep things easier to maintain.
- The GyroAccel class has been deprecated, and is replaced bij the new OrientationSensors class
- Magnetometer has been added to the list of input sensors.
- It's now possible to (automatically) set the game north to the real world compass north.
- The magnetometer can be used to compensate for gyro drift (still experimental, though!)
- The VirtualTouchpad, VirtualJoystick and VirtualButton classes have all been deprecated, and have been replaced bij the new TouchInterface class
- UI controls are now always drawn directly to the screen, using unity's superfast low-level graphics library (GL).
- UI controls can now also be referenced by their unique ID number (int), instead of using their names (string). Less work for the garbage collector.
- The option to use SendMessage, to automatically and constantly send messages to gameobjects, has been deprecated, due to it's rather slow performance. Instead, directly use the API in your scripts.
- Makeover of the inspector windows, making configurations even easier.
- New demo scenes, significantly decreasing package size.

v5.1
- IMPORTANT: as always, first make a backup of your project, before updating.
- Improved compatibility for older devices that only support accelerometers.
- GetRoll() will now return a clamped result, between -180 and 180 degrees, just like it's GetHeading() and GetPitch() counterparts already did.
- New Api commands GetHeadingUnclamped() and GetRollUnclamped().
- Fixed a bug in autoUpdate, where a clamped roll could suddenly jump from one boundary to another.
- AutoUpdate now supports the option to 'push the viewports edge', when a heading or pitch tries to move beyond it's allowed boundaries.
- GyroAccel, VirtualJoysticks, VirtualTouchpads and VirtualButtons now all support the ability to use the keyboard and mouse to simulate movements of the gyro, joysticks, etc. No more need to always use Unity Remote to test your application ;-)

v5.0
- IMPORTANT: as always, first make a backup of your project, before updating.
- Unity 5.x only
- Heading and pitch can now be forced to a certain value, by using the new API commands SetHeading() and SetPitch(). Finally an easy way to (re)set the rotation of your player character.
- AutoUpdating the orientation of GameObjects now allows you to set boundaries, to keep values within a certain limit. Also, when one chooses not to autoUpdate a certain rotation axis, a fixed value can be chosen (previously, this value would be zero by default).
- Each GameObject in the autoUpdate list can have it's own smoothingTime values.
- OnGUI is only being used while editing the scene. During runtime, all VirtualButtons, VirtualJoysticks and VirtualTouchpads are now directly drawn on screen, to increase drawing speed.
- VirtualButtons, VirtualJoysticks and VirtualTouchpads can now easily be activated or deactivated during runtime, using their new API commands Enable() and Disable().
- VirtualButtons and VirtualJoysticks can now be resized and repositioned during runtime, using their new API commands SetSize() and SetCenter().
- VirtualTouchpads can now be resized and repositioned during runtime, using their new API command SetRect().
- Double tapping VirtualJoysticks and VirtualTouchpads can now be checked by calling their new API commands GetDoubleTap() and GetDoubleTapHold().
- All new demo scenes, based on the sample scenes available from Unity.
- The FPS rigidbody player prefab in the demo scenes has been replaced by a more advanced version. See demo 1a and 1b.
- Various small bug fixes.

v4.6
- IMPORTANT: as always, first make a backup of your project, before updating.
- Configuration of the GyroAccel, VirtualJoystick, VirtualTouchpad and VirtualButtons scripts have greatly improved: There is a specialized custom inspector window for each of them. Customize the gyro's settings and/or create a joystick, touchpad and button with just a view mouse clicks and your done. Al changes are directly visible in the gameview, without running the game.
- Due to the new configuration tools, large parts of the code have been rewritten. The package is backwards compatible, with a few exceptions: The possibility to create a VirtualJoystick, VirtualTouchpad or VirtualButton during runtime has been removed from the API.
- Also the MSP_CharacterMotor.cs script has been removed. It has been replaced by a must more simplistic script, based on RigidBodies. This allows for a better understanding how the input controls can be integrated into your own character movement scripts.
- The package has been tested with and updated for Unity 5.0. The third demo scene (Simple Car Driving Game) still shows some jittering, due to the fact that Unity's 5.x wheelcolliders are not fully compatible with Unity 4.x wheelcolliders. This will be fixed in a future update.
- Please note that this will be the last update for Unity 4.x. All future updates will be Unity 5.x only!

v4.1
- Moved some functionality from FixedUpdate() to Update()
- VirtualButtons can now also be used as a switch
- Fixed a 'missing component'-error in one of the prefabs - fixed swapped naming error of two prefabs ("dual-joystick" <--> "joystick+touchpad") v4.0.5
- Fixed an error in MSP_CharacterMotor:IsTouchingCeiling()
- Fixed an error where a VirtualTouchpad would occasionally not update correctly while using a VirtualJoystick

v4.0
- All new version!!
- Completely rewritten in c#
- Gyro/accel / virtual joysticks / virtual touchpads / virtual buttons are easily configurable in the inspector
- Many new commands, to get/set various settings and variables.
- IMPORTANT: Version 4.x is not compatible with older versions: Please update your existing projects before updating]

v3.1
- For all those people who haven't yet upgraded to Unity 4.x: The asset has been made backwards compatable with Unity Version 3.5.7f6
- Few typo's fixed

v3.0.x
- Now compatible with Unity4.0
- More comment lines added in the scripts
- Readme.txt added with 'how-to-use' instructions
- Code has been partly rewritten and optimized

Legal Notice