DESCRIPTION

This asset enables you to read the gyroscope and accelerometer information provided by your browser. The main uses of this are VR-like 360 experiences and AR experiences. I created this asset because I wanted to use the gyroscope for a client project but it isn't supported in WebGL. It's very flexible and easy to use.

FEATURES

1. Enables you to read the gyroscope and accelerometer.
2. You can detect if you successfully accessed the gyroscope/accelerometer or not. All the errors that can happen in the JS side are transferred to Unity, so you can manage them.
3. Works on Safari, Chrome, Firefox, and Opera.

HOW DOES IT WORK?

It uses the devicemotion api for the accelerometer, and the deviceorientation api for the gyroscope. It exposes both apis completely, through the *deviceMotionEvent* and *deviceOrientationEvent* variables in *MotionSensorsWebGL.cs*, respectively. These variables are supposed to be read from only, you shouldn't write to them. Here are their types:

```
public struct DeviceMotionEvent {
        public struct Acceleration {public float x, y, z;}
        public struct AccelerationIncludingGravity {public float x, y, z;}
        public struct RotationRate {public float alpha, beta, gamma;}
        public Acceleration acceleration;
        public AccelerationIncludingGravity accelerationIncludingGravity;
        public RotationRate rotationRate;
        public float interval;
}

public struct DeviceOrientationEvent {
        public float alpha;
        public float beta;
        public float gamma;
        public float webkitCompassHeading;
        public float webkitCompassAccuracy;
        public float x;
```

```
        public float y;
        public float z;
        public float w;
        public bool absolute;
}
```

As you may have noticed, they are exactly like their javascript counterparts, with a few differences:

1. In javascript, those variables are represented as doubles, not floats. Unity uses floats for everything, so I changed that.
2. Another difference that you may have noticed, is on the *DeviceOrientationEvent*. It has 4 additional variables: *x,y,z,w*. These are the quaternion representations of the euler angles *alpha*, *beta* and *gamma*.

The raw rotation provided(*alpha*, *beta* and *gamma*) isn't immediately usable. For one, Unity uses quaternions instead of eulers to update the rotation of objects. Secondly, simply calling Quaternion.Euler does not convert it to a quaternion that works in the way you expect. Some additional black magic math needs to be done to get the appropriate quaternion values. I did just that to save you the trouble. In practice, this means you will always want to read *x,y,z,w* instead of *alpha,beta,gamma*.

HOW DO I USE IT?

Attach the *MotionSensorsWebGL.cs* script to any gameobject. You are only supposed to have one in the scene. There is only one field in the inspector, the *permissionRequest*. When using the gyroscope/accelerometer, the browser can make a permission request to access the hardware. Additional checks to see if the hardware is sending data and the browser exposes the API are also done. The result of all this is informed in a *string* (a json string specifically) that is sent to the callbacks that you add in the inspector.

Possible results are:

1. *"\"name\":\"PermissionState\",\"message\":\"Granted\""* -> sent when permission is granted and hardware is ready to use.

2. *"\"name\":\"NotAllowedError\",\"message\":\"Requesting device orientation or motion access requires a user gesture to prompt\""* -> sent when trying to access hardware without user interaction(clicking on a button). As of the time of writing, Safari is the only browser that can throw this error.

3. *"\"name\":\"PermissionState\",\"message\":\"Denied\""* -> User explicitly denied hardware access. As of the time of writing, Safari is the only browser that can throw this error.

4. *"\"name\":\"DeviceNotSendingDataError\",\"message\":\"Data not being sent. Gyroscope/Accelerometer might not be present, malfunctioning, or disabled. Is this a desktop?\""* -> custom error that I created when the hardware data can't be read, even though there weren't permission issues. This typically happens on desktop browsers because they don't have gyroscope/accelerometer.

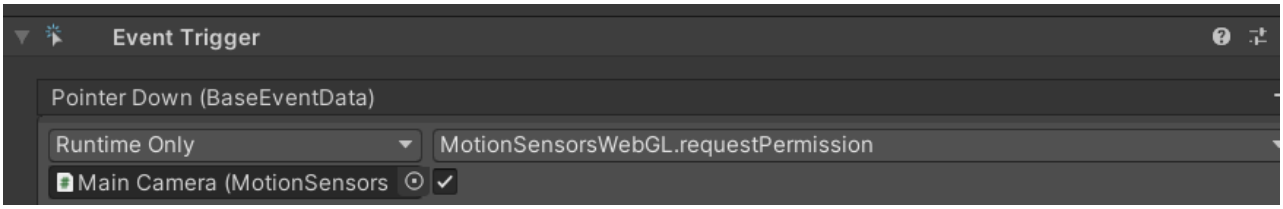
5. *"\"name\":\"UnsupportedAPIError\",\"message\":\"Your browser does not support the \'DeviceOrientationEvent\' and/or \'DeviceMotionEvent\' APIs\""* -> custom error that I created that is thrown when the browser doesn't support the API to access the accelerometer/gyroscope from the device.

As for the methods:

1. *public void registerDevicemotionEvent()* -> registers the devicemotion event in the javascript side to read the accelerometer data. Call this method if you want to use the accelerometer.
2. *public void registerDeviceorientationEvent()* -> registers the deviceorientation event in the javascript side to read the gyroscope data. Call this method if you want to use the gyroscope.
3. *public void unregisterDevicemotionEvent()* -> unregisters the devicemotion event. If you want to stop using the accelerometer, call this method.
4. *public void unregisterDeviceorientationEvent()* -> unregisters the deviceorientation event. If you want to stop using the gyroscope, call this method.
5. *public bool isRequestPermissionEverRequired()* -> checks if requesting permission to use the gyroscope/accelerometer is required at least once(after the user grants permission, the browser can cache the result and not request again after the user reloads the page). As of the time of writing, only Safari requires permission to be granted. Returns *true* if permission is ever required, *false* if not.
6. *public void requestPermission(bool userInteraction = false)* -> Requests permission to use the gyroscope/accelerometer, and also checks if the hardware is sending data and the browser supports the api. You should pass false to the parameter if you want to request permission **without** user interaction. As of the time of writing, this will work on all browsers except Safari. You should pass true if you want to request permission **with** user interaction. This will work on all browsers, including Safari. If you pass *true* to the parameter you must call this method on a pointerdown event. You should always call this method, even if permission is not required, because of the hardware and API checks.

The *RotateWebGL.cs* is an example script that you can use to rotate anything in a VR-like manner(typically the camera). You can use the target field or leave it empty, in which case the target will become the gameobject it is attached to. Check out the example scene, the button to request permission is only enabled if requesting permission is necessary in the first place. The way it knows this is by attempting to request permission and checking if the *NotAllowedError* is thrown. The code is heavily commented on.

Please note on the example scene, when calling the *requestPermission* method on the *PermissionRequest* button, the *userInteraction* parameter is *true* and the method is called on a *pointerdown* event:



Restrictions

You need to upload your build to an **https(secure http)** server. Using http:// or file://(when you simply open the html file on your computer by double clicking it) **won't work**! I recommend using https-localhost or netlify. This is not a restriction with my plugin, it's just how browsers work.

A note about Firefox

As of the time of writing, sometimes firefox acts like Chrome, without needing a button to request permission, and sometimes it acts like Safari, needing a button to request permission. The thing is, it always acts as if it doesn't need permission. It doesn't throw any errors if you don't use a button, but it can fail to work. If you need to support firefox, it's best to always have the button to request permission enabled. Firefox will only work consistently with a button.