Sensors For The Web!

By Alexander Shalamov
Alex is an Engineer at Intel

By Mikhail Pozdnyakov
Mikhail is an Engineer at Intel

Today, sensor data is used in many native applications to enable use cases such as immersive gaming, fitness tracking, and augmented or virtual reality. Wouldn't it be cool to bridge the gap between native and web applications? The Generic Sensor API, For The Web!

What is Generic Sensor API?

The Generic Sensor API is a set of interfaces which expose sensor devices to the web platform. The API consists of the base Sensor interface and a set of concrete sensor classes built on top. Having a base interface simplifies the implementation and specification process for the concrete sensor classes. For instance, take a look at the Gyroscope class, it is super tiny! The core functionality is specified by the base interface, and Gyroscope merely extends it with three attributes representing angular velocity.

Typically a concrete sensor class represents an actual sensor on the platform e.g., accelerometer or gyroscope. However, in some cases, implementation of a sensor class fuses data from several platform sensors and exposes the result in a convenient way to the user. For example, the AbsoluteOrientation sensor provides a ready-to-use 4x4 rotation matrix based on the data obtained from the accelerometer, gyroscope and magnetometer.

You might think that the web platform already provides sensor data and you are absolutely right! For instance, DeviceMotion and DeviceOrientation events expose motion sensor data, while other experimental APIs provide data from an environmental sensors. So why do we need new API?

Comparing to the existing interfaces, Generic Sensor API provides great number of advantages:

- Generic Sensor API is a sensor framework that can be easily extended with new sensor classes and each of these classes will keep the generic interface. The client code written for one sensor type can be reused for another one with very few modifications!
- You can configure sensor, for example, set the sampling frequency suitable for your application needs.

- You can detect whether a sensor is available on the platform.
- Sensor readings have high precision timestamps, enabling better synchronization with other activities in your application.
- Sensor data models and coordinate systems are clearly defined, allowing browser vendors to implement interoperable solutions.
- The Generic Sensor based interfaces are not bound to the DOM (neither Navigator nor Window objects), and it opens up future opportunities of using the same API within service workers or implementing Generic Sensor API in headless JS runtimes, for instance, on embedded devices.
- Security and privacy aspects are the top priority for the Generic Sensor API and provide much better security level compared to older sensor APIs. There is integration with Permissions API.
- Automatic synchronization with screen coordinates is available for Accelerometer, Gyroscope, LinearAccelerationSensor, AbsoluteOrientationSensor, RelativeOrientationSensor and Magnetometer.

Generic Sensor APIs in Chrome

At the time of writing, Chrome supports several sensors that you can experiment with.

Motion sensors:

- Accelerometer
- Gyroscope
- LinearAccelerationSensor
- AbsoluteOrientationSensor
- RelativeOrientationSensor

Environmental sensors:

- AmbientLightSensor
- Magnetometer

The Generic Sensor-based motion sensor classes are enabled by default in Chrome starting from Chrome 67. Use the chrome://flags/#enable-generic-sensor feature flag to enable motion sensors on the previous Chrome versions.

You can enable environmental sensors for development purposes by turning on a feature flag. Go to chrome://flags/#enable-generic-sensor-extra-classes to enable environmental sensors. Restart Chrome and you should be good to go.

More information on browser implementation status can be found on chromestatus.com

What are all these sensors? How can I use them?

Sensors is a quite specific area which might need a brief introduction. If you are familiar with sensors, you can jump right to the hands-on coding section. Otherwise, let's look at each supported sensor in detail.

Accelerometer and linear acceleration sensor

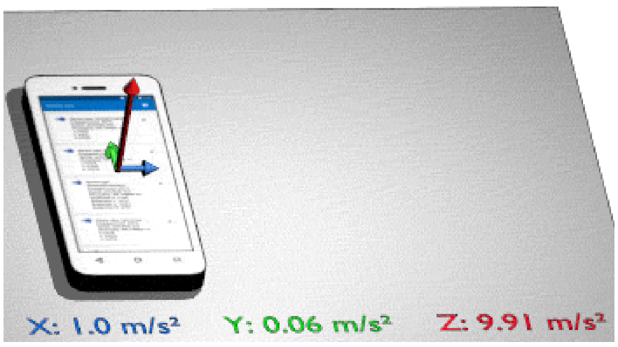

Figure 1: Accelerometer sensor measurements

The Accelerometer sensor measures acceleration of a device hosting the sensor on three axes (X, Y and Z). This sensor is an inertial sensor, meaning that when the device is in linear free fall, the total measured acceleration would be 0 m/s^2, and when a device lying flat on a table, the acceleration in upwards direction (Z axis) will be equal to the Earth's gravity, i.e. g ≈ +9.8 m/s^2 as it is measuring the force of the table pushing the device upwards. If you push device to the right, acceleration on X axis would be positive, or negative if device is accelerated from right toward the left.

Accelerometers can be used for things like: step counting, motion sensing or simple device orientation. Quite often, accelerometer measurements are combined with data from other sources in order to create fusion sensors, such as, orientation sensors.

The LinearAccelerationSensor measures acceleration that is applied to the device hosting the sensor, excluding the contribution of a gravity force. When a device is at rest, for instance, lying flat on the table, the sensor would measure ≈ 0 m/s^2 acceleration on three axes.

Gyroscope

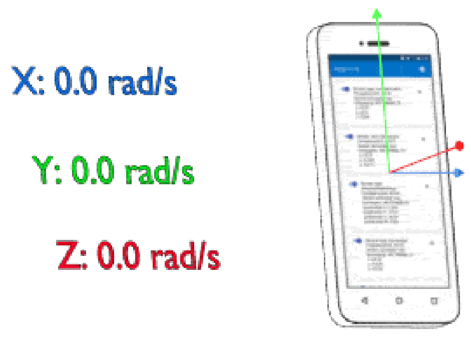

Figure 2: Gyroscope sensor measurements

The Gyroscope sensor measures angular velocity in rad/s around the device's local X, Y and Z axis. Most of the consumer devices have mechanical (MEMS) gyroscopes, which are inertial sensors that measure rotation rate based on inertial Coriolis force. A MEMS gyroscopes are prone to drift that is caused by sensor's gravitational sensitivity which deforms the sensor's internal mechanical system. Gyroscopes oscillate at relative high frequencies, e.g., 10's of kHz, and therefore, might consume more power compared to other sensors.

Orientation sensors

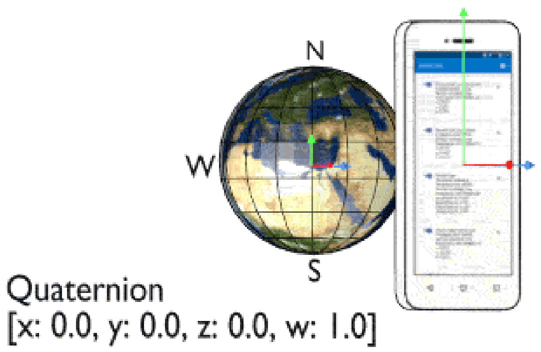

Figure 3: AbsoluteOrientation sensor measurements

The AbsoluteOrientationSensor is a fusion sensor that measures rotation of a device in relation to the Earth's coordinate system, while the RelativeOrientationSensor provides data representing rotation of a device hosting motion sensors in relation to a stationary reference coordinate system.

All modern 3D JavaScript frameworks support quaternions and rotation matrices to represent rotation; however, if you use WebGL directly, the OrientationSensor interface has convenient methods for WebGL compatible rotation matrices. Here are few snippets:

three.js

```
let torusGeometry = new THREE.TorusGeometry(7, 1.6, 4, 3, 6.3);
let material = new THREE.MeshBasicMaterial({ color: 0x0071C5 });
let torus = new THREE.Mesh(torusGeometry, material);
scene.add(torus);

// Update mesh rotation using quaternion.
const sensorAbs = new AbsoluteOrientationSensor();
sensorAbs.onreading = () => torus.quaternion.fromArray(sensorAbs.quaternion);
sensorAbs.start();

// Update mesh rotation using rotation matrix.
const sensorRel = new RelativeOrientationSensor();
let rotationMatrix = new Float32Array(16);
sensor_rel.onreading = () => {
    sensorRel.populateMatrix(rotationMatrix);
    torus.matrix.fromArray(rotationMatrix);
}
sensorRel.start();
```

BABYLON

```
const mesh = new BABYLON.Mesh.CreateCylinder("mesh", 0.9, 0.3, 0.6, 9, 1, s
const sensorRel = new RelativeOrientationSensor({frequency: 30});
sensorRel.onreading = () => mesh.rotationQuaternion.FromArray(sensorRel.quaternio
sensorRel.start();
```

WebGL

```
// Initialize sensor and update model matrix when new reading is available.
let modMatrix = new Float32Array([1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1]);
const sensorAbs = new AbsoluteOrientationSensor({frequency: 60});
sensorAbs.onreading = () => sensorAbs.populateMatrix(modMatrix);
sensorAbs.start();

// Somewhere in rendering code, update vertex shader attribute for the model
gl.uniformMatrix4fv(modMatrixAttr, false, modMatrix);
```

Orientation sensors enable various use cases, such as immersive gaming, augmented and virtual reality.

For more information about motion sensors, advanced use cases, and requirements, please check motion sensors explainer document.

Synchronization with screen coordinates

By default, spatial sensors' readings are resolved in a local coordinate system that is bound to the device and does not take screen orientation into account.

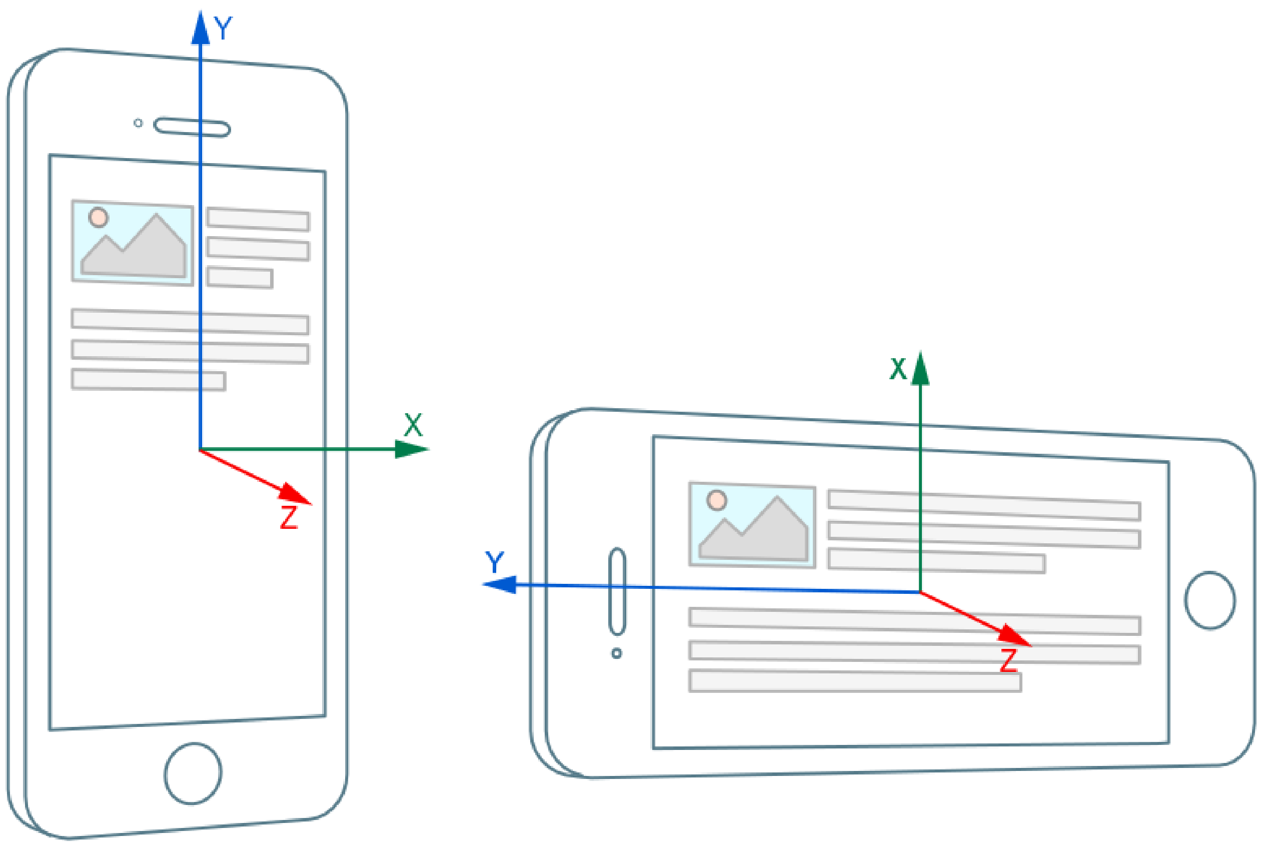

Figure 4: Device coordinate system

However, many use cases like games or augmented and virtual reality require sensor readings to be resolved in a coordinate system that is instead bound to the screen orientation.

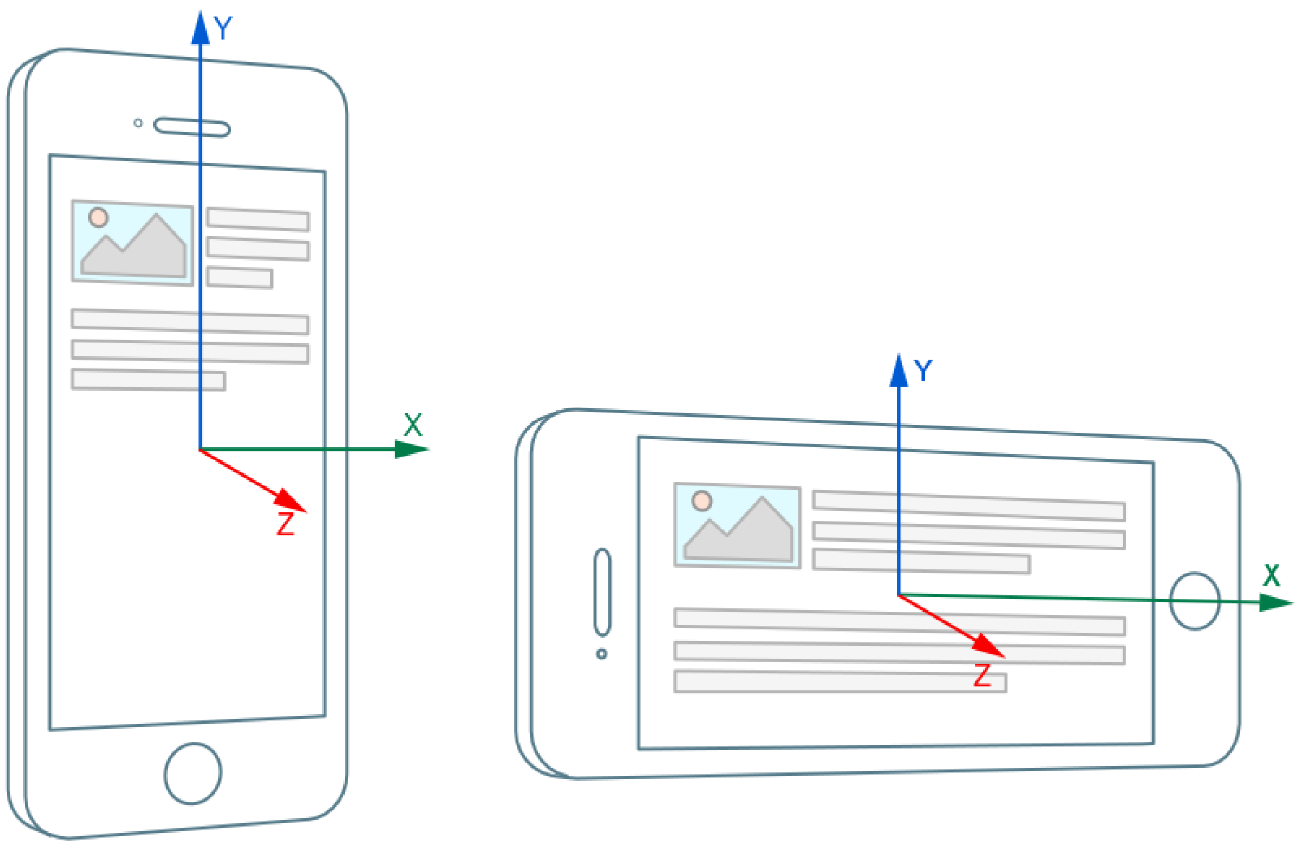

Figure 5: Screen coordinate system

Previously, remapping of sensor readings to screen coordinates had to be implemented in JavaScript. This approach is inefficient and it also quite significantly increases the complexity of the web application code: the web application must watch screen orientation changes and perform coordinates transformations for sensor readings, which is not a trivial thing to do for Euler angles or quaternions.

The Generic Sensor API provides much simpler and reliable solution! The local coordinate system is configurable for all defined spatial sensor classes: Accelerometer, Gyroscope, LinearAccelerationSensor, AbsoluteOrientationSensor, RelativeOrientationSensor and Magnetometer. By passing the **`referenceFrame`** option to the sensor object constructor the user defines whether the returned readings will be resolved in device or screen coordinates.

```
// Sensor readings are resolved in the Device coordinate system by default.
// Alternatively, could be RelativeOrientationSensor({referenceFrame: "device"}).
const sensorRelDevice = new RelativeOrientationSensor();

// Sensor readings are resolved in the Screen coordinate system. No manual remapp
const sensorRelScreen = new RelativeOrientationSensor({referenceFrame: "screen"})
```

Note: The `referenceFrame` sensor option is supported in Chrome 66 or later.

Let’s code!

The Generic Sensor API is very simple and easy-to-use! The Sensor interface has **`start()`** and **`stop()`** methods to control sensor state and several event handlers for receiving notifications about sensor activation, errors and newly available readings. The concrete sensor classes usually add their specific reading attributes to the base class.

Development environment

During development you'll be able to use sensors through **`localhost`**. The simplest way is to serve your web application using Web Server for Chrome. If you are developing for mobile devices, set up port forwarding for your local server, and you are ready to rock!

When your code is ready, deploy it on a server that supports HTTPS. GitHub Pages are served over HTTPS, making it a great place to share your demos.

Note: Don't forget to enable Generic Sensor API in Chrome versions prior to Chrome 67.

3D model rotation

In this simple example, we use the data from an absolute orientation sensor to modify the rotation quaternion of a 3D model. The **`model`** is a three.js **`Object3D`** class instance that has a **`quaternion`** property. The following code snippet from the orientation phone demo, illustrates how the absolute orientation sensor can be used to rotate a 3D model.

```
function initSensor() {
    sensor = new AbsoluteOrientationSensor({frequency: 60});
    sensor.onreading = () => model.quaternion.fromArray(sensor.quaternion);
    sensor.onerror = event => {
        if (event.error.name == 'NotReadableError') {
            console.log("Sensor is not available.");
        }
    }
    sensor.start();
}
```

The device's orientation will be reflected in 3D **`model`** rotation within the WebGL scene.

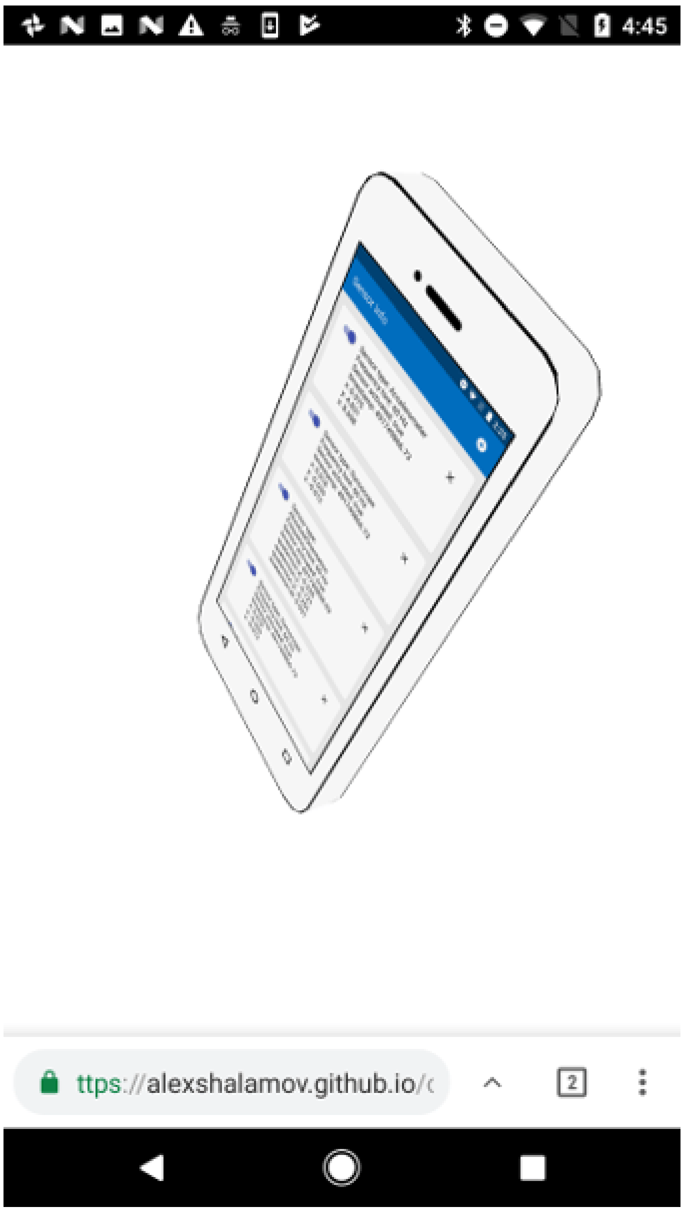

Figure 6: Sensor updates orientation of a 3D model

Punchmeter

The following code snippet is extracted from the punchmeter demo, illustrating how the linear acceleration sensor can be used to calculate the maximum velocity of a device under the assumption that it is initially laying still.

```
this.maxSpeed = 0;
this.vx = 0;
this.ax = 0;
this.t = 0;

function onreading() {
    let dt = (this.accel.timestamp - this.t) * 0.001; // In seconds.
    this.vx += (this.accel.x + this.ax) / 2 * dt;
    let speed = Math.abs(this.vx);

    if (this.maxSpeed < speed) {
        this.maxSpeed = speed;
    }

    this.t = this.accel.timestamp;
    this.ax = this.accel.x;
}

....

this.accel.addEventListener('reading', onreading);
```

The current velocity is calculated as an approximation to the integral of the acceleration function.

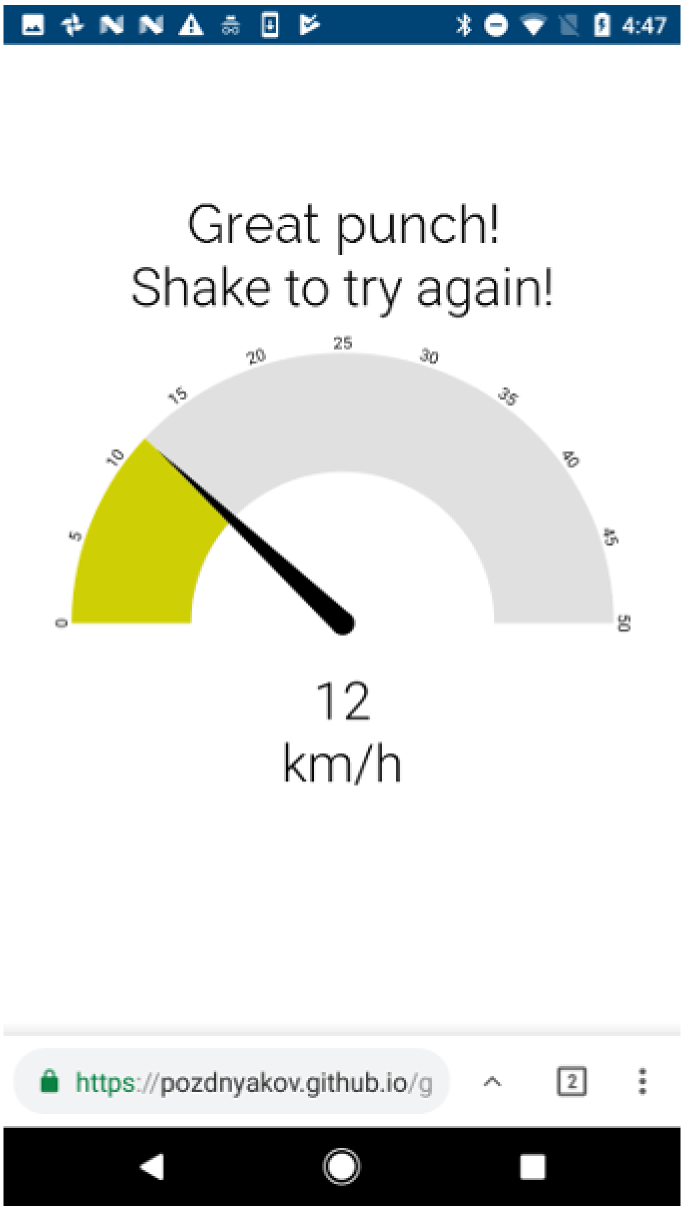

Figure 7: Measurement of a punch speed

Privacy and security

Sensor readings are sensitive data which can be subject to various attacks from malicious web pages. Chrome's implementation of Generic Sensor APIs enforces few limitations to mitigate the possible security and privacy risks. These limitations must be taken into account by developers who intend to use the API, so let's briefly list them.

Only HTTPS

Because Generic Sensor API is a powerful feature, Chrome only allows it on secure contexts. In practice it means that to use Generic Sensor API you'll need to access your page through HTTPS. During development you can do so via http://localhost but for production you'll need to have HTTPS on your server. See Security with HTTPS article for best practices and guidelines there.

Feature Policy integration

The Feature Policy integration in Generic Sensor API is used to control access to sensors data for a frame.

By default the `Sensor` objects can be created only within a main frame or same-origin subframes, thus preventing cross-origin iframes from unsanctioned reading of sensor data. This default behavior can be modified by explicitly enabling or disabling of the corresponding policy-controlled features.

The snippet below illustrates granting accelerometer data access to a cross-origin iframe, meaning that now `Accelerometer` or `LinearAccelerationSensor` objects can be created there.

```
<iframe src="https://third-party.com" allow="accelerometer"/>
```

Note: The Feature Policy integration for sensors is available in Chrome 65 or later. In the earlier versions of Chrome the `Sensor` objects can be created only within a main frame.

Sensor readings delivery can be suspended

Sensor readings are only accessible by a visible web page, i.e., when the user is actually interacting with it. Moreover, sensor data would not be provided to the parent frame if the user focuses to a cross-origin subframe in order to disallow the parent frame infer user input.

What’s next?

There is a set of already specified sensor classes to be implemented in the near future such as Proximity sensor and Gravity sensor; however thanks to the great extensibility of Generic Sensor framework we can anticipate appearance of even more new classes representing various sensor types.

Another important area for future work is improving of the Generic Sensor API itself, the Generic Sensor specification is currently a draft which means that there is still time to make fixes and bring new functionality that developers need.

You can help!

The sensor specifications reached Candidate Recommendation maturity level, hence, the feedback from web and browser developers is highly appreciated. Let us know what features would be great to add or if there is something you would like to modify in the current API.

Please fill the survey. Also feel free to file specification issues as well as bugs for the Chrome implementation.

Resources

- Demo projects: https://intel.github.io/generic-sensor-demos/
- Generic Sensor API specification: https://w3c.github.io/sensors/
- Specification issues: https://github.com/w3c/sensors/issues
- W3C working group mailing list: public-device-apis@w3.org
- Chrome Feature Status: https://www.chromestatus.com/feature/5698781827825664
- Implementation Bugs: http://crbug.com?q=component:Blink>Sensor

- Sensors-dev Google group: https://groups.google.com/a/chromium.org/forum/#!forum/sensors-dev