# Hands-on

https://www.w3.org/TR/generic-sensor/

# Generic Sensor API

## W3C Candidate Recommendation, 12 December 2019

**This version:**
https://www.w3.org/TR/2019/CR-generic-sensor-20191212/

**Latest published version:**
https://www.w3.org/TR/generic-sensor/

**Editor's Draft:**
https://w3c.github.io/sensors/

**Previous Versions:**
https://www.w3.org/TR/2019/WD-generic-sensor-20190307/

**Feedback:**
public-device-apis@w3.org with subject line "`[generic-sensor] … message topic …`" (archives)
GitHub (new issue, level 1 issues, all issues)

**Editor:**
Rick Waldron (Bocoup, formerly on behalf of JS Foundation)

**Former Editors:**
Mikhail Pozdnyakov (Intel Corporation)
Alexander Shalamov (Intel Corporation)
Tobie Langel (Codespeaks, formerly on behalf of Intel Corporation) tobie@codespeaks.com

**Test Facilitator:**
Wanming Lin (Intel Corporation)

**Other:**
Test suite, latest version history, previous version history



---

## Abstract

This specification defines a framework for exposing sensor data to the Open Web Platform in a consistent way. It does so by defining a blueprint for writing specifications of concrete sensors along with an abstract Sensor interface that can be extended to accommodate different sensor types.

# Scope

- Specifying primitives which enable exposing data from device sensors.
- Exposing remote sensors or sensors found on personal area networks (e.g. Bluetooth) is out of scope.

# Detect HW

```
if (typeof Gyroscope === "function") {
    // run in circles...
}

if ("ProximitySensor" in window) {
    // watch out!
}

if (window.AmbientLightSensor) {
    // go dark...
}

// etc.
```

Presence and  characteristics of an API, no info on  API
1) actually connected to  real hardware
2) whether it works,
3) if its still connected,
4) you can access it.

# Note

- Why not info on  underlying status  available upfront.
  - Getting this information out of the hardware is costly, in both performance and battery time, and would sit in the critical path.
  - The status of the underlying hardware can evolve over time. The user can revoke permission, the connection to the sensor be severed, the operating system may decide to limit sensor usage below a certain battery threshold, etc.

```
if (typeof Gyroscope === "function") {
    // run in circles...
}

if ("ProximitySensor" in window) {
    // watch out!
}

if (window.AmbientLightSensor) {
    // go dark...
}

// etc.
```

Combine feature detection, which checks whether an API for the sought-after sensor actually exists, and defensive programming

```
let accelerometer = null;
try {
    accelerometer = new Accelerometer({ frequency: 10 });
    accelerometer.addEventListener('error', event => {
        // Handle runtime errors.
        if (event.error.name === 'NotAllowedError') {
            console.log('Permission to access sensor was denied.');
        } else if (event.error.name === 'NotReadableError' ) {
            console.log('Cannot connect to the sensor.');
        }
    });
    accelerometer.addEventListener('reading', () => reloadOnShake(accelerometer));
    accelerometer.start();
} catch (error) {
    // Handle construction errors.
    if (error.name === 'SecurityError') {
        console.log('Sensor construction was blocked by the Feature Policy.');
    } else if (error.name === 'ReferenceError') {
        console.log('Sensor is not supported by the User Agent.');
    } else {
        throw error;
    }
}
```

1. checking for error thrown when instantiating a Sensor object,

2. listening to errors emitted by it,

3. handling all of the above graciously so that the user’s experience is enhanced by the possible usage of a sensor, not degraded by its absence.

# Security and privacy considerations

- Sensor readings are sensitive data and could become a subject of various attacks from malicious Web pages.
- The risk of successful attack can increase when
  - Multiple sensors/functions are used (correlation)
    - Minimize accuracy
  - Used over time (fingerprinting)
    - Minimize sampling time

# Threats

- Location Tracking: use sensor readings to locate the device without using GPS or any other location sensors.
  - For example, accelerometer data can be used to infer the location of smartphones by using statistical models to obtain estimated trajectory, then map matching algorithms can be used to obtain predicted location points (within a 200-m radius)

# Threats

- Eavesdropping: Recovering speech from gyroscope readings
- Keystroke Monitoring: many user inputs can be inferred from sensor readings, this includes a wide range of attacks on user PINs, passwords, and lock patterns (and even touch actions such as click, scroll, and zoom) using motion sensors. These attacks normally train a machine learning algorithm to discover such information about the users.

# Threats

- Device Fingerprinting: Sensors can provide information that can uniquely identify the device using those sensors. Every concrete sensor model has minor manufacturing imperfections and differences that will be unique for this model. These manufacturing variations and imperfections can be used to fingerprint the device
- User Identifying: Sensor readings can be used to identify the user, for example via inferring individual walking patterns from smartphone or wearable device motion sensors' data.

# Mitigation Strategies

- Secure Context: Sensor Readings (SR) are explicitly flagged by the Secure Contexts specification
- Feature Policy: SR are only available for the documents which are allowed to use the policy-controlled features for the given sensor type.

# Mitigation Strategies

- Focused Area: SR are only available for active documents whose origin is same origin-domain with the currently focused area document.
- Visibility State: SR are only available for the active documents whose visibility state is "visible".
- Permissions API: SR are controlled by the Permissions API

# Mitigation Strategies

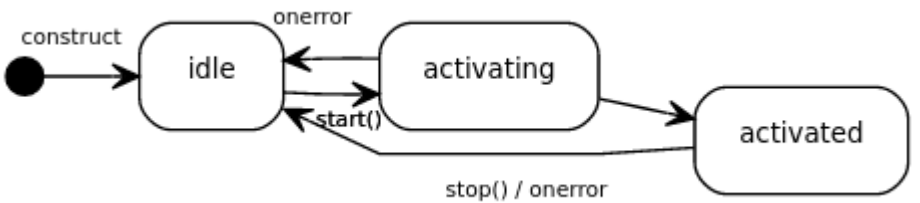
Main risks due to correlation, fingerprinting

- Limit maximum sampling frequency
- Stop the sensor altogether
- Limit number of delivered readings
- Reduce accuracy
- Keep the user informed about API use

# Sensor Interface

```
[SecureContext, Exposed=(DedicatedWorker, Window)]
interface Sensor : EventTarget {
  readonly attribute boolean activated;
  readonly attribute boolean hasReading;
  readonly attribute DOMHighResTimeStamp? timestamp;
  void start();
  void stop();
  attribute EventHandler onreading;
  attribute EventHandler onactivate;
  attribute EventHandler onerror;
};

dictionary SensorOptions {
  double frequency;
};
```



https://intel.github.io/generic-sensor-demos/

# Generic Sensor API playground

Here you can find list of web applications based on Generic Sensor APIs

View on GitHub

## Demos for Generic Sensor API

This repository contains applications that demonstrate how to use the 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, such as Accelerometer, LinearAccelerationSensor, Gyroscope, AbsoluteOrientationSensor and RelativeOrientationSensor.

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.

# FIRST TEST

https://intel.github.io/generic-sensor-demos/sensor-info/build/bundled/

We've created a set of resources to help you ensure your site remains available and accessible to all during the COVID-19 situation.

Home > Products > Web > Updates > By Year

# 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?

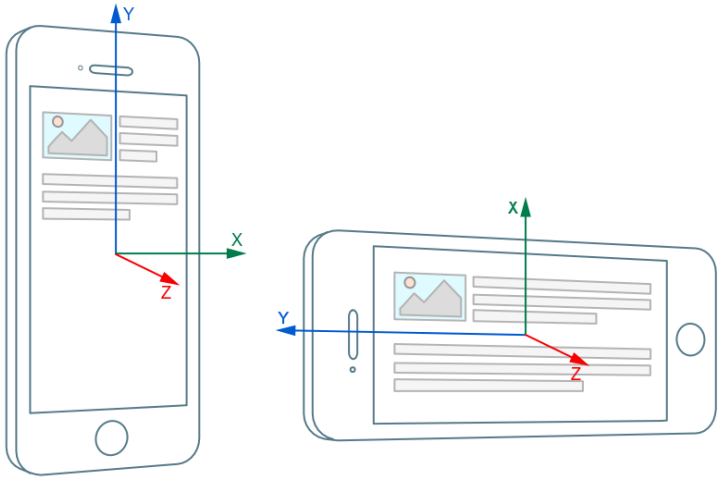
The Generic Sensor API is a set of interfaces which expose sensor devices to the web platform. The API consists of the base Sensor interface and a set of concrete sensor classes built on top. Having a base interface simplifies the implementation and specification process for the concrete sensor classes. 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.

Contents



Nice description on sensors

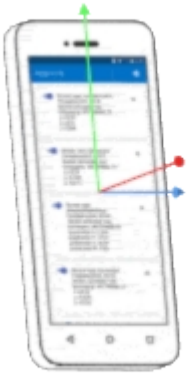
https://developers.google.com/web/updates/2017/09/sensors-for-the-web

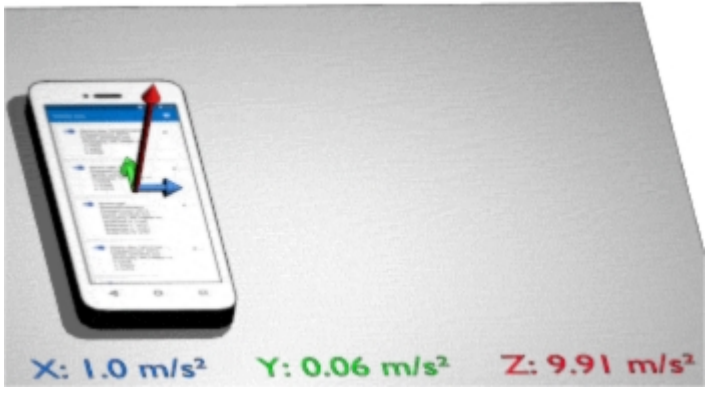


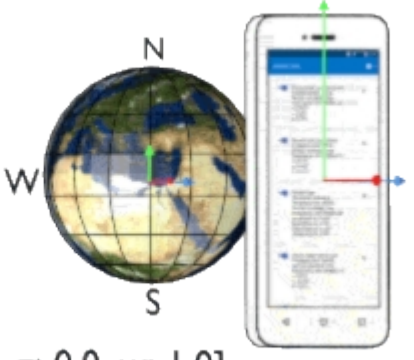

X: 0.0 rad/s

Y: 0.0 rad/s

Z: 0.0 rad/s





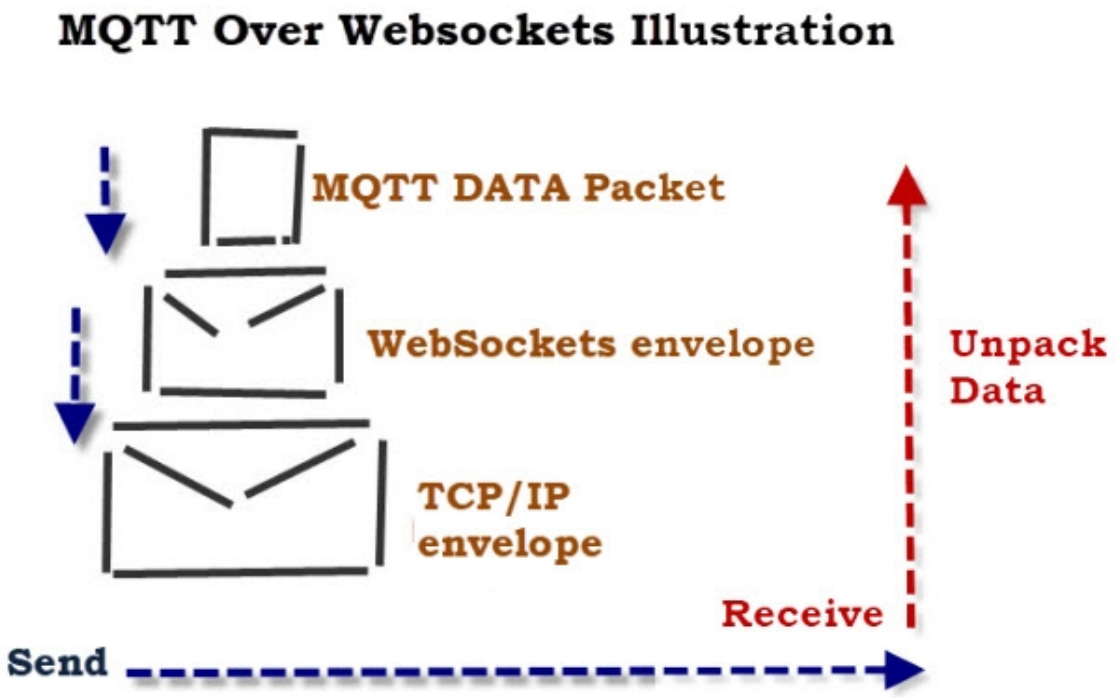

Quaternion
[x: 0.0, y: 0.0, z: 0.0, w: 1.0]

# Delivering data

- Note that in principle we are talking about a huge amount of data
  - Edge computing
  - Privacy
- However a simplistic assumption coherent with the course
  - MQTT

# MQTT on the front-end

- MQTT over websockets



```
// Create a client instance
client = new Paho.MQTT.Client("127.0.0.1",9001, "clientId");

// set callback handlers
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;

// connect the client
client.connect({onSuccess:onConnect});


// called when the client connects
function onConnect() {
  // Once a connection has been made, make a subscription and send a message.
  console.log("onConnect");
  client.subscribe("World");
  message = new Paho.MQTT.Message("Hello");
  message.destinationName = "World";
  client.send(message);
}

// called when the client loses its connection
function onConnectionLost(responseObject) {
  if (responseObject.errorCode !== 0) {
    console.log("onConnectionLost:"+responseObject.errorMessage);
  }
}

// called when a message arrives
function onMessageArrived(message) {
  console.log("onMessageArrived:"+message.payloadString);
}
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible"
content="IE=edge">
    <title>Hello MQTT World</title>
    <meta name="viewport"
content="width=device-width, initial-scale=1">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/paho-
mqtt/1.0.1/mqttws31.min.js"></script>
    <script src="main.js" defer></script>
</head>
<body>
    <div id="logger"></div>
</body>
</html>
```

http-server