SensFloor® Live API

The SensFloor® API allows you to build your own application by taking advantage of the different levels of pre-processed data it provides. For instance, you may wish to know if there's any activity on the floor at all; or count the number of persons in the area; or trigger an alarm when a fall is detected.

Data from the API are provided in the form of events. While some events are triggered asynchronously (ASYNC) when a SensFloor® message is received by the SE10 or some other condition is met, the majority is generated synchronously, roughly around every 100ms (SYNC).

App Version

SensFloor Live App - **v3.2.0**

How to use

The API is reachable via web sockets, namely via socket.io (see https://socket.io/). Socket.io clients exist for Node.js and for the browser in JavaScript. Furthermore, other implementations in other languages are available (see https://github.com/socketio/socket.io).

To use to the API, you need to know the URL of the SE10. In case you connected via Wi-Fi, the URL shall be: http://192.168.5.5:8000. If you connected via Ethernet, the URL is dependent on your IP, for instance: http://192.168.100.105:8000. See the *Setup* section for more details.

Using JavaScript on the browser, you can connect to the API and get data as follows:

```
<script src="http://192.168.5.5:8000/socket.io/socket.io.js"></script>
<script>
        var client = io.connect('http://192.168.5.5:8000');
        client.on('connect', function() {
                console.log('connected');
        });

        client.on('messages-raw', function(messages) {
                for (var i = 0; i < messages.length; i++) {
                        console.log(messages[i]);
                }
        });
</script>
```

Events (consume)

There are several events that can be accessed through the SensFloor® API, depending on the abstraction level you need to build your own application. These events are either triggered by SensFloor® messages or specific conditions (ASYNC), or periodically emitted roughly every 100ms by the system (SYNC).

To use data from each event, simply register a listener for the event name and retrieve the data (See the *Examples* sub-section for more details).

'messages-raw'

Description

This event sends out every SensFloor® message received, without any processing. It pre-appends a timestamp to the array, in milliseconds since the epoch.

Type

ASYNC

Data

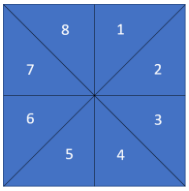
Array of time-stamped full SensFloor messages.

Format

```
[
        [TIME, BYTE0, BYTE1, ..., BYTE15],
        [TIME, BYTE0, BYTE1, ..., BYTE15],
        ...
]
```

TIME	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15
TIME	GID0	GID1	ID0	ID1	DC	DC	DC	DC	FIELD1	FIELD2	FIELD3	FIELD4	FIELD5	FIELD6	FIELD7	FIELD8

Time: Milliseconds since the epoch
GID1,0: Two bytes comprising the Group ID of the installation
ID0,1: Two bytes comprising the ID of the SensFloor® patch
DC: Don't care
FIELD1…8: "Capacitance" value for each sensor field, between 128 and 255



Data example

```
[ [1476180138377, 3, 65, 7, 2, 179, 0, 17, 2, 175, 142, 128, 127, 128, 127, 132, 130] ]
```

'new-activity-on-fields'

Description

This event sends out messages every time at least one of the sensor fields has a new capacitance value. It can be used to update the capacitance of individual fields in your application, instead of the whole patch at one time.

Type
ASYNC

Data
Array of time-stamped new capacitance values per field.

Format
[
    [TIME, MID, FIELD, CAP],
    [TIME, MID, FIELD, CAP],
    ...
]

Data example
```
[ [1476189894888, 2818572, 1, 129] ]
```

'step'

Description
This event is triggered every time a new step is detected on SensFloor® and provides information about the location of the step. Step detection performance is dependent on gait and floor resolution.

Type
ASYNC

Data
Two numbers containing the location of the step.

Format
x, y

Data example
```
0.398, 0.667
```

'activity-update'

Description
This event sends out information about which points (centroids of the sensor fields) changed capacitance and state roughly in the last 100 milliseconds.

A point (field) changing state means that its capacitance crossed a certain threshold (set to 140 by default). That is, a point can either be active or inactive.

Type
SYNC

Data
An object containing information on which points changed state or capacitance value since the last sampling period.

Format
{

flags: {
newCapacitance,
newState
},
newCapacitance,
newState

}

.flags.newCapacitance [boolean] - true if at least one point has a new capacitance value.
.flags.newState [boolean] - true if at least one point has a new state (ON/OFF = Active/Inactive).
.newCapacitance [array] - points that have a new capacitance value.
point {
.**id**: number of the point
.**capacitance**: value of the field capacitance [127 – 255]
.**x**: world position on the x axis, in meters
.**y**: world position on the y axis, in meters
}
.newState [array] - points that have a new state.
point {
.**id**: number of the point
.**isActive**: true if the point is active (True/False)
.**birthTime**: When the point became active, in milliseconds since epoch
.**x**: world position on the x axis, in meters
.**y**: world position on the y axis, in meters
}

Data examples
- Point 4 becomes active

```
activity: {
    flags: {
        newCapacitance: true,
        newState: true
    },
    newCapacitance: [ {id: 4, capacitance: 150, x: 0.167, y: 0.102} ],
    newState: [ {id: 4, isActive: true, birthTime: 1476372959279, x: 0.167, y: 0.102} ]

}
```

- Point 12 has a new capacitance value but did not change state

```
activity: {
    flags: {
        newCapacitance: true,
        newState: false
    },
    newCapacitance: [ {id: 12, capacitance: 176, x: 0.667, y: 0.102} ],
    newState: []
```

```
}
```

‘clusters-update’

Description

This event provides information about clusters of active points (centroids of sensor fields). Each cluster is comprised of at least one point and has a unique ID assigned to it. In simple terms, active points close to each other, that became active within a short period of time, form a cluster. Conversely, two points that are either far away from each other, or became active at very distinct times, will form two separate clusters.

Type

SYNC

Data

An array containing the current clusters.

Format

[cluster, cluster, ...]

```
cluster: {
      id,
      birthTime,
      x,
      y,
      xWeighted,
      yWeighted,
      age,
      weight,
      size,
      hull,
      area,
      spread
}
```

.id [number] – cluster id.
.birthTime [number] - when the cluster was created, in milliseconds since epoch.
.**x** [number] – cluster world position on the x axis, in meters.
.**y** [number] – cluster world position on the y axis, in meters.
.**xWeigthed** [number] - capacitance-weighted position on the x axis, in meters.
.**yWeigthed** [number] - capacitance-weighted position on the y axis, in meters.
.**age** [number] – time since the cluster was created, in milliseconds.
.**weight** [number] - average capacitance value of the cluster's points.
.**size** [number] - amount of points belonging to the cluster.
.**hull** [array] - array of vertices (x, y) that define the convex hull of the cluster.
.**area** [number] - area of the convex hull, in square meters.
.**spread** [number] - cluster points' standard deviation, in meters.

Data example

Array with two clusters

```
[ { id: 0,
    birthTime: 1479827359404,
    x: 0.7310000000000001,
    y: 0.7310000000000001,
    xWeighted: 0.7310000000000001,
    yWeighted: 0.7310000000000001,
    averageBirthTime: 1479827359460,
    age: 11337,
    weight: 150,
    size: 2,
    hull: [ [Object], [Object] ],
    area: 0,
    spread: 0.132 },
{   id: 1,
    birthTime: 1479827365235,
    x: 2.5989999999999998,
    y: 1.247,
    xWeighted: 2.5989999999999998,
    yWeighted: 1.247,
    averageBirthTime: 1479827365266,
    age: 5506,
    weight: 150,
    size: 3,
    hull: [ [Object], [Object], [Object] ],
    area: 0.02177999999999991,
    spread: 0.2498719672152121 } ]
```


‘objects-update’

Description

The objects event provides information about the objects (normally associated to persons) that are detected on SensFloor®. Objects are born from groups of clusters. The location of an object is estimated from the current location of its clusters and a simple movement prediction system. Furthermore, each object keeps track of how many steps were detected in its vicinity.

Type

SYNC

Data

An array containing the current objects.

Data Format

[object, object, ...]

```
object: {
      id,
      birthTime,
      birthX,
      birthY,
      x,
      y,
```

```
        age,
        mV,
        aV,
        steps
}
```

.id [number] – object id.
.birthTime [number] - when the object was created, in milliseconds since epoch.
.**birthX** [number] – world position of where the object was created on the x axis, in meters.
.**birthX** [number] – world position of where the object was created on the y axis, in meters.
.**x** [number] – object world position on the x axis, in meters.
.**y** [number] – object world position on the y axis, in meters.
.**age** [number] – time since the object was created, in milliseconds.
.**mV** [number] – magnitude component of the velocity vector for the object, in meters per second.
.**aV** [number] – angle component of the velocity vector for the object, in degrees.
.**steps** [number] – number of steps detected near the object.

Data
An array containing the current objects.

Data example
Array with one object

```
[ { id: 0,
    birthTime: 1479828662153,
    birthX: 0.7310000000000001,
    birthY: 0.7310000000000001,
    x: 1.124036889454593,
    y: 1.2421012866769736,
    age: 2100,
    mV: 0.3223749077759499,
    aV: 52.439715720681185,
    steps: 2 } ]
```

‘falls-detected’ – DEPRECATED (use alarm events instead)

Description
This event contains information about falls detected on SensFloor®. It provides details about the fall, such as its location, the polygon that circumscribes the area of the fall, and an Oriented Bounding Box (OBB) for that polygon.

Type
SYNC

Data
Two arrays containing the current falls. The first array contains information about the bounding boxes. The second about the convex hulls.

Data Format
[OBBInfo, OBBInfo, …], [hullsInfo, hullsInfo, …]

```
OBBInfo: {
      points,
```

```
        centroid
}
```

.points [array] – the four vertices defining the bounding box for the fall.
    point[array]: world position of the vertex, in meters [x, y].
.centroid [array] – world position of the box centroid, in meters [x, y].

```
hullsInfo: [points, centroid]
```

points [array] – the polygon vertices defining the convex hull of the fall.
    point[array]: world position of the vertex, in meters [x, y].
centroid [array] – world position of the convex hull centroid, in meters [x, y].

Data example
One fall detected.

```
[ { points: [ [Object], [Object], [Object], [Object] ],
    centroid: [ 0.605, 0.434 ]
} ]

[ [ [ [Object], [Object], [Object], [Object], [Object] ],
    [ 0.589, 0.484 ] ]
]
```

'relays-state'

Description
This event emits the state of the eight physical relays of the SE10 every time it changes (e.g., when an alarm is triggered). A value of true means the associated relay is closed. Refer to the SE10 datasheet for more info about the relays.

Type
ASYNC

Data
An array of booleans containing the relays' state.

Format
[r0, r1, r2, r3, r4, r5, r6, r7]

Data example
Relays 1 and 3 are closed

```
[0, 1, 0, 1, 0, 0, 0, 0]
```

'alarms-detected'

Description
This event generates the alarms that are currently detected. Please note that all alarms defined through the SensFloor Config App will be included here if detected by the algorithm(s). This can mean alarms that are defined but: disabled; outside the time window; configured to not generate API

events.
If you wish to consume only alarms that are 'active', please refer to the 'alarms-active' event.

Type
SYNC

Data
An array of detected alarms.

Format
[alarm, alarm, ...]

```
alarm: {
       index,
       type,
       state,
       activationTime,
       activationLocation,
       active,
       event,
}
```

.index [number] – alarm index (as defined in the Config App).
.type [string] – alarm type (Presence, Movement, Count, Fall, Toilet, Toilet In, Toilet Out, Toilet Timer, Bed In, Bed Out, Bed Timer, Room In, Room Out, External Sensor).
.state [boolean] – whether alarm is detected (should always be true in this event).
.activationTime [number] – when the alarm was first detected, in milliseconds since epoch.
.activationLocation [object] – world position of the alarm, in meters {x, y}.
.active [boolean] – whether the alarm is active when detected.
.event [boolean] – whether the alarm should generate an API event (or only switch the relays).

Data examples
"Presence" alarm detected


```
[{
"index": 0,
"type": "presence",
"state": true,
"activationTime": 1628583965624,
"activationLocation": { "x": 5.668, "y": 1.102 },
"active": true,
"event": true

}]
```


"Room In" alarm detected but disabled


```
[{
"index": 2,
"type": "room in",
"state": true,
"activationTime": 1628583961224,
"activationLocation": { "x": 2.1, "y": 0.4 },
"active": false,
"event": true

}]
```

‘alarms-active’

Description

This event generates the alarms that are currently detected and active.

Type

SYNC

Data

An array of active alarms.

Format

[alarm, alarm, ...]

```
alarm: {
      index,
      type,
      state,
      activationTime,
      activationLocation
}
```

.index [number] – alarm index (as defined in the Config App).
.type [string] – alarm type (Presence, Movement, Count, Fall, Toilet, Toilet In, Toilet Out, Toilet Timer, Bed In, Bed Out, Bed Timer, Room In, Room Out, External Sensor).
.state [boolean] – whether the alarm is detected (should always be true in this event).
.activationTime [number] – when the alarm was first detected, in milliseconds since epoch.
.activationLocation [object] – world position of the alarm, in meters {x, y}.

Data example

“Fall” alarm active

```
[{

"index": 1,
"type": "fall",
"state": true,
"activationTime": 1628583965624,
"activationLocation": { "x": 1.123, "y": 3.224 },

}]
```

‘alarms-changed’

Description

This event triggers when an alarm changes state. If you only care about consuming alarm changes (for active alarms), you should use this event.

Type

ASYNC

Data

An array of active alarms that changed detected state.

Format

[alarm, alarm, ...]

```
alarm: {
       alarmNumber,
       type,
       state,
       time,
       location
}
```

.alarmNumber [number] – alarm number in 1-based index. (this corresponds to the alarm index + 1).
.type [string] – alarm type (Presence, Movement, Count, Fall, Toilet, Toilet In, Toilet Out, Toilet Timer, Bed In, Bed Out, Bed Timer, Room In, Room Out, External Sensor).
.state [boolean] – whether the alarm changed to detected (true) or back to not-detected (false).
.time [number] – when the alarm changed state.
.location [object] – world position of the alarm, in meters {x, y}. This will NOT change for the not-detected event. For instance, a 'Presence' alarm changing to not-detected (person left the floor) will still include the location where the person first stepped on the floor.

Data examples

"Toilet In" alarm start event


```
[{

"alarmNumber": 4,
"type": "toilet in",
"state": true,
"time": 1628583911124,
"location": { "x": 3.668, "y": 4.122 }

}]
```


"Toilet In" alarm end event


```
[{

"alarmNumber": 4,
"type": "toilet in",
"state": false,
"time": 1628583916124,
"location": { "x": 3.668, "y": 4.122 }

}]
```

Examples

This section provides you with three examples of how the SensFloor® API can be access through JavaScript in the browser.

To see each example, create an empty HTML file, paste the code you see below, and save the file (e.g. Example1.html). Afterwards, open the file you just created in your browser of choice and you will be able to see different information when you walk or fall on top of SensFloor®. Make sure you are connected to the SE10, either via Wi-Fi or via LAN. For the latter case, change the two IP addresses on each example to match the SE10's IP address on your local network.

Example 1 – Printing raw data

```
<!DOCTYPE html>
<html>
    <head>
        <title>SensFloor - Example 1 - Raw Data</title>
        <script src="http://192.168.5.5:8000/socket.io/socket.io.js"></script>
    </head>
    <body>
        <textarea id=logger></textarea>
        <script>
            var client = io.connect('http://192.168.5.5:8000');

            client.on('connect', function() {
                console.log('connected');
            });

            client.on('messages-raw', function(messages) {
                for (var i = 0; i < messages.length; i++) {
                    var date = new Date(messages[i][0]);
                    var timeStamp = date.getFullYear() + '-' +
                        ('0' + (date.getMonth() + 1)).slice(-2) + '-' +
                        ('0' + date.getDate()).slice(-2) + '-' +
                        ('0' + date.getHours()).slice(-2) + ':' +
                        ('0' + date.getMinutes()).slice(-2) + ':' +
                        ('0' + date.getSeconds()).slice(-2) + ':' +
                        ('00' + date.getMilliseconds()).slice(-3);

                    var messageString = timeStamp;
                    for (b = 1; b <= 16; b++) {
                        var hexByte = ('0' + (messages[i][b] &
                         0xFF).toString(16)).slice(-2).toUpperCase();
                        messageString += ' ' + hexByte;
                    }
                    var logger = document.getElementById('logger');
                    logger.innerHTML += messageString + '&#010;';
                    logger.scrollTop = logger.scrollHeight;
                }
```

```
            });
        </script>
        <style>
            body {
                background-color: lightgrey;
            }
            #logger {
                width: 90%;
                height: 90%;
                top: 5%;
                left: 5%;
                position: absolute;
                resize: none;
                font-size: 2.25vw;
            }
        </style>
    </body>
</html>
```

Example 2 – Tracking persons and speed

```
<!DOCTYPE html>
<html>
    <head>
        <title>SensFloor - Example 2 - Tracking persons and speed</title>
        <script src="http://192.168.5.5:8000/socket.io/socket.io.js"></script>
    </head>
    <body>
        <div id=number-persons></div>
        <div id=static-persons></div>
        <div id=moving-persons></div>
        <div id=max-speed></div>
        <div id=speed-record></div>
        <script>
            var client = io.connect('http://192.168.5.5:8000');
            var speedRecord = 0;

            client.on('connect', function() {
                console.log('connected');
            });

            client.on('objects-update', function(objects) {
                var nPersons = nStanding = nWalking = maxSpeed = 0;
                for (var i = 0; i < objects.length; i++) {
                    nPersons++;
                    if (objects[i].mV > 0) {
                        nWalking++
                    }
                    else {
```

```
                    nStanding++;
                }
                if (objects[i].mV > maxSpeed) {
                    maxSpeed = objects[i].mV;
                }
                if (objects[i].mV > speedRecord) {
                    speedRecord = objects[i].mV;
                }
            }
            document.getElementById('number-persons').innerHTML = nPersons + ' Person(s)';
            document.getElementById('static-persons').innerHTML = nStanding + ' Standing';
            document.getElementById('moving-persons').innerHTML = nWalking +  ' Walking';
            document.getElementById('max-speed').innerHTML = 'Max Speed: ' +
            (maxSpeed * 3.6).toFixed(1) + ' km/h';
            document.getElementById('speed-record').innerHTML = 'Speed Record: ' +
            (speedRecord * 3.6).toFixed(1) + ' km/h';
        })
    </script>
    <style>
        div {
            width: 40%;
            height: 20%;
            position: absolute;
            font-size: 4vw;
            border: 1px solid black;
            display: flex;
            align-items: center;
            justify-content: center;
            background-color: lightgrey;
        }
        #number-persons {
            top: 5%;
            left: 5%;
            width: 90%;
            height: 25%;
            font-size: 10vw;
        }
        #static-persons {
            left: 5%;
            top: 40%;
        }
        #moving-persons {
            left: 55%;
            top: 40%;
        }
        #max-speed {
            left: 5%;
            top: 70%;
```

```
                color: blue;
            }
            #speed-record {
                left: 55%;
                top: 70%;
                color: red;
            }
        </style>
    </body>
</html>
```

Example 3 – Detecting falls

```
<!DOCTYPE html>
<html>
    <head>
        <title>SensFloor - Example 3 - Detecting falls</title>
        <script src="http://192.168.5.5:8000/socket.io/socket.io.js"></script>
    </head>
    <body>
        <div id=current-fall></div>
        <div id=last-fall>No previous falls detected</div>
        <script>
            var client = io.connect('http://192.168.5.5:8000');
            var lastFallDate = '';
            var lastFallTime = '';

            client.on('connect', function() {
                console.log('connected');
            });

            client.on('alarms-active', function(alarms) {
                var activeFall = false;
                for (var i = 0; i < alarms.length; i++) {
                    if (alarms[i].type === 'fall') {
                        var date = new Date(alarms[i].activationTime);
                        var fallDate =  date.getFullYear() + '-' +
                                        ('0' + (date.getMonth() + 1)).slice(-2) + '-' +
                                        ('0' + date.getDate()).slice(-2);
                        var fallTime =  ('0' + date.getHours()).slice(-2) + ':' +
                                        ('0' + date.getMinutes()).slice(-2) + ':' +
                                        ('0' + date.getSeconds()).slice(-2);
                        document.getElementById('current-fall').innerHTML = 'FALL DETECTED !';
                        document.getElementById('last-fall').innerHTML =
                                        'Last fall detected on ' + fallDate + ' at ' + fallTime;
                        document.getElementById('current-fall').style.color = "red";
                        document.getElementById('last-fall').style.color = "red";
                        document.body.style.background = 'red';
                        activeFall = true;
```

```
                        break;
                    }
                }

                if (!activeFall) {
                    document.getElementById('current-fall').innerHTML = 'No falls';
                    document.getElementById('current-fall').style.color = "black";
                    document.body.style.background = 'none';
                }
        });
        </script>
        <style>
            div {
                width: 90%;
                left: 5%;
                position: absolute;
                border: 1px solid black;
                display: flex;
                align-items: center;
                justify-content: center;
                background-color: lightgrey;
            }
            #current-fall {
                height: 50%;
                top: 5%;
                font-size: 10vw;
            }
            #last-fall {
                height: 25%;
                top: 70%;
                font-size: 4.5vw;
            }
        </style>
    </body>
</html>
```