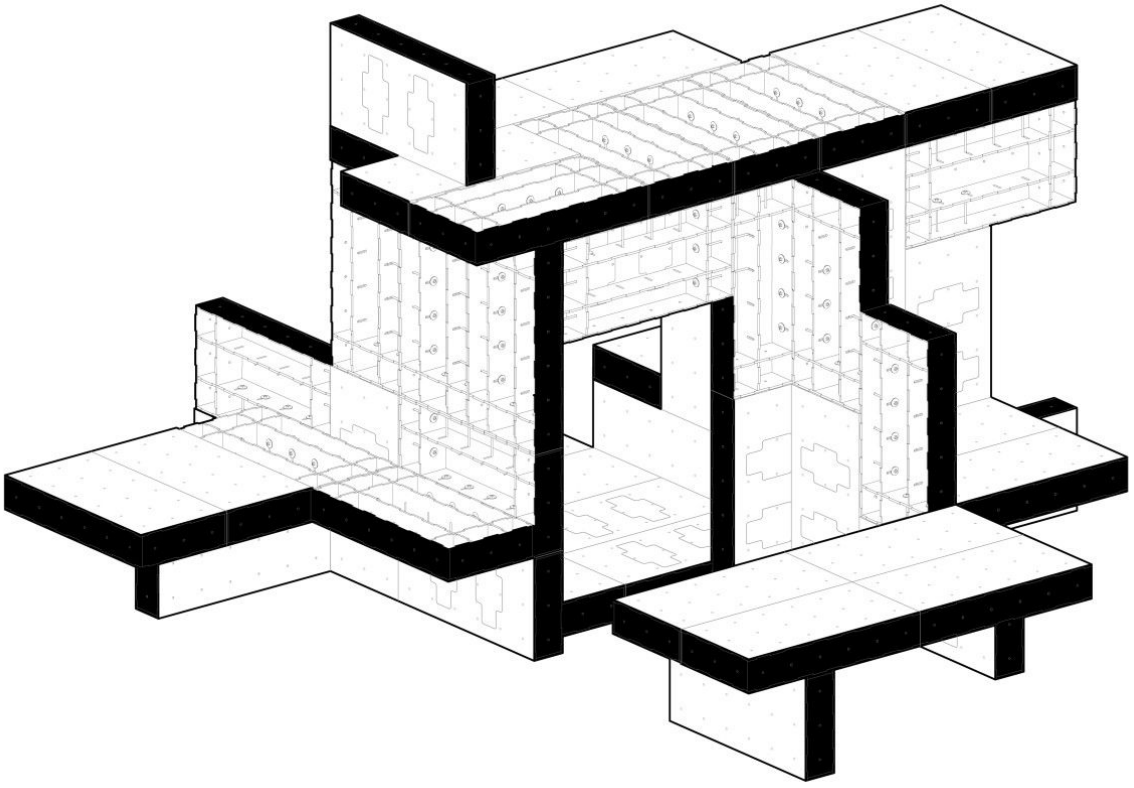
Xsens DOT Server
System Architecture



Version:	1.0
Date:	March 29, 2020
Author:	Fred Dijkstra
Doc. id:	rtaoqbv

Document history

Version	Date	Author(s)	Description
1.0	March 29, 2020	Fred Dijkstra (fred@oryxmovementsolutions.nl)	First version

Table of contents

Introduction

This document describes the system architecture of the *Xsens DOT Server*, a system for recording the orientation data of BLE enabled motion trackers.

The system is built using Node.js and therefore programmed in JavaScript. By doing so, the application is platform independent, but the deployment as done in this project was done on a Raspberry Pi 3B.

This system architecture document identifies the system components and gives a short description on their roles, context and interfacing. Documentation is available to describe the detailed design of the implemented components.

System components

The functionality of the system is divided into a number of identifiable components. The diagram is given in figure 1.

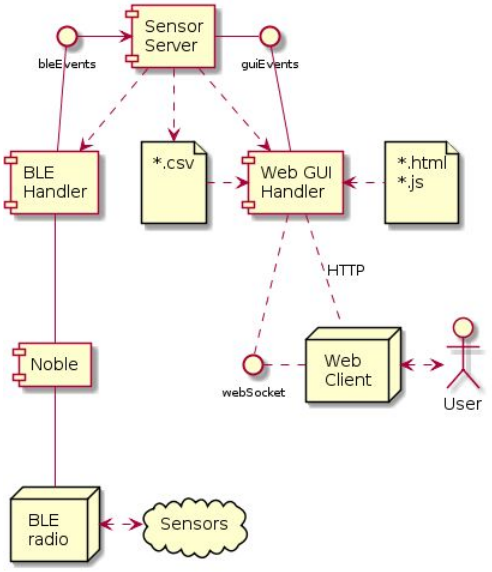


Figure 1: System components diagram.

A short description of each identified system component and the documentation (if relevant) are given in table 1.

Table 1: Short description of the system components.

Component	Description	Doc. id.
Sensor Server	Application and workflow control.	wq4ywkl
BLE Handler	Creates an abstraction from the BLE protocol.	rwtej5x
*.csv	Comma separated values files that are generated as a result of a recording.	-
Web GUI Handler	The web server.	rtaoqbv
*.html, *.js	HTML and JavaScript files required by the web page.	-
Noble	Node package that implements an interface with the BLE radio (i.e. driver).	GitHub repo
Web Client	A web browser that can run on any computer on the local network and that renders an HTML page that implements the GUI.	v4zyyhm
User	The person interfacing with the web page and thereby controlling the workflow.	-
BLE radio	The combination of the driver and transceiver to implement BLE.	-
Sensors	The orientation tracking sensors that communicate via BLE.	-

Components

BLE Handler

Role

The BLE Handler creates an abstract interface to a network of BLE enabled orientation sensors and takes care of all the specifics of the BLE protocol implementation. Practically this also means that when different BLE sensors are used, only the BLE Handler needs to be adjusted.

Context

The context of the BLE Handler is given in figure 2 and is formed by the Sensor Server and the Noble object.

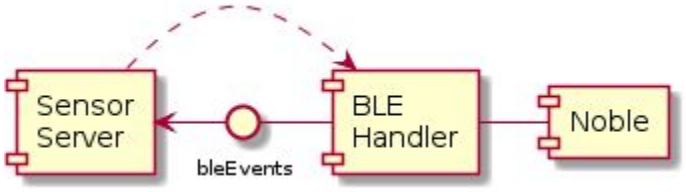


Figure 2: Context of the BLE Handler.

Interfacing

The interface of the BLE Handler comprises interface functions and events that are handled by the **`eventHandler`** callback function specified as the argument of the constructor.

The interface functions of the BLE Handler (i.e. its methods) are given in table 11.

Table 2: BLE Handler interface functions.

Method	Arguments	Description
`constructor`	• `eventHandler`	Constructs a new BLE Handler that will instantiate a Noble object in order to start using the BLE radio. The eventHandler argument is a callback function that points to a function in the Sensor Server to have relevant events handled by the Sensor Server.
`startScanning`	• -	The scanning for new sensors can be started.
`stopScanning`	• -	Scanning for new sensors can be stopped.
`connectSensor`	• `sensor`	The connection to the specified sensor can be established.
`disconnectSensor`	• `sensor`	Disconnect the specified sensor.
`enableSensor`	• `sensor`	Enable the measurements on the specified sensor.
`disableSensor`	• `sensor`	Disable the measurements on the specified sensor.

The specified `eventHandler()` function has an event `name` and the `parameters` object as arguments. Whenever an event occurs at the BLE Handler that is to be converted into an event for the Sensor Server, the `eventHandler()` callback function is called. The possible BLE Events are listed in table 3.

Table 3: BLE Events as generated by the BLE Handler.

Event name	Parameters	Description
`blePoweredOn`	• `-`	The BLE radio is ready to be used.
`bleScanningStarted`	• `-`	Scanning was successfully started.
`bleScanningStopped`	• `-`	Scanning stopped.
`bleSensorDiscovered`	• `sensor`	A new sensor is discovered.
`bleSensorConnected`	• `sensor`	The sensor is connected.
`bleSensorDisconnected`	• `sensor`	The sensor was disconnected.
`bleSensorEnabled`	• `sensor`	The sensor is measuring and sensor data can be expected.
`bleSensorDisabled`	• `sensor`	The sensor is no longer measuring.
`bleSensorData`	• `timestamp` • `address` • `q_w` • `q_x` • `q_y` • `q_z`	A new orientation was received for the sensor. The orientation is expressed as a normalized quaternion.
`bleSensorError`	• `sensor` • `error`	

The properties of the sensor parameter that are relevant for the Sensor Server are given in table 4.

Table 4: Relevant properties of the sensor parameter.

Property	Type	Description
`name`	String	The name as specified by the sensor when advertising.
`address`	String	A unique MAC address by which the sensor can be uniquely identified.

Noble

Role

Noble is a Node.js package[1] that can be used to create a platform independent interface to the actual implementation of the BLE driver.

It implements the central BLE role and as such offers the functionality to scan for peripherals, establish connections to discovered peripherals, perform Services Discovery and access the characteristics of the services hosted by a peripheral.

Context

The context in which the Noble module is used is given in figure 3 and is formed by the BLE Handler and the BLE radio which implements the physical interface to the sensor network.

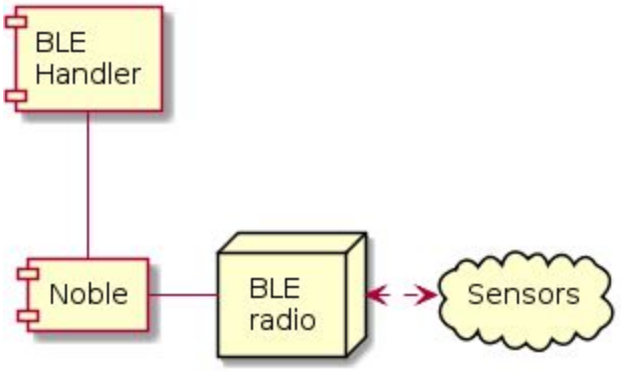


Figure 3: Context of the usage of the Noble module.

Interfacing

The Noble module offers interfacing on 3 levels: central, peripheral and characteristic. These interfaces comprise functions and events.

The interface functions of the central object are given in table 5.

Table 5: Noble central interface functions.

Method	Arguments	Description
`startScanning`	• `services` • `allowDuplicates`	Starts the scanning for peripherals. The **`services`** argument is an array containing the UUIDs of the Services that the peripheral must advertise in order for it to be discovered. The **`allowDuplicates`** argument indicates whether duplicates are allowed, i.e. whether the advertising packet must be processed each time it is received.
`stopScanning`	• -	Stop scanning for peripherals.

[1] https://github.com/noble/noble

The events that can be generated by the central object are given in table 6.

Table 6: Events as generated by the Central object.

Event name	Parameters	Description
`statusChange`	• `state`	When the state is "*poweredOn*", the BLE radio is ready to be used.
`scanStart`	• `-`	Scanning was started.
`scanStop`	• `-`	Scanning stopped.
`discover`	• `peripheral`	A new peripheral was discovered, i.e. its advertisement packet was received.

The relevant properties of the **`peripheral`** parameter of the "*discover*" event are given in table 7.

Table 7: Relevant properties of the peripheral object.

Property	Type	Description
`address`	String	MAC address of the BLE transceiver on the peripheral.
`advertisement`	Object	Object containing the data as was received in the advertising packet. The relevant property used in this system is the `localName`.

The relevant interface functions for the peripheral are given in table 8.

Table 8: Noble peripheral interface functions.

Method	Arguments	Description
`connect`	• `callback(error)`	Try to connect to the peripheral.
`disconnect`	• `callback(error)`	Disconnect from the peripheral.
`discoverAllServicesAndCharacteristics`	• `callback(error, services[], characteristics[])`	Perform a Services Discovery to retrieve all services and characteristics hosted by the peripheral.

The relevant events that can be generated by the peripheral object are given in table 9.

Table 9: Events as generated by the peripheral object.

Event name	Parameters	Description
`connect`	• `-`	The peripheral is connected.
`disconnect`	• `-`	The peripheral disconnected.

As the result of a Services Discovery, a list of characteristics is returned in the callback function. The relevant interface functions for a characteristic are given in table 10.

Table 10: Noble characteristic interface functions.

Method	Arguments	Description
`write`	• `data` • `withoutResponse` • `callback(error)`	Writes the `data` (buffer) to the characteristic. The `withoutResponse` argument is set to `false` when this is an "authenticated write".
`subscribe`	• -	Enable the notifications for the characteristic.

The relevant events that can be generated by the characteristic object are given in table 9.

Table 9: Events as generated by the characteristic object.

Event name	Parameters	Description
`data`	• `data` • `isNotification`	The value of the characteristic as a result of the notification (`isNotification` = true).

Sensor Server

Role

The Sensor Server implements the behavior and workflow of the application. This means that when this needs to be adjusted, only the Sensor Server needs to be changed.

At an abstract level the Sensor Server can be considered as the interface between the GUI and the sensor network.

The Sensor Server also takes care of recording the sensor data into a CSV file which can then be accessed by the Web GUI Handler to offer it for downloading.

Context

The context of the Sensor Server is given in figure 4 and is formed by the BLE Handler and the Web GUI Handler.

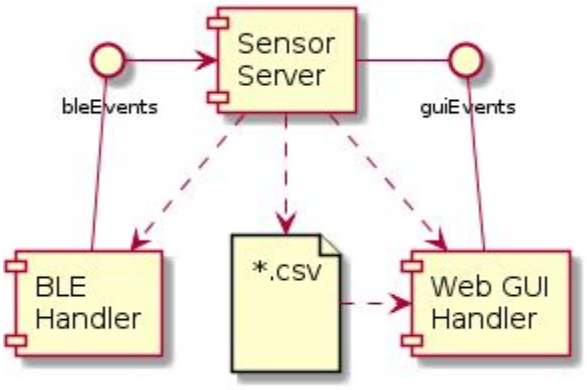


Figure 4: Context of the Sensor Server.

Interfacing

The Sensor Server uses the interfaces as supplied by the BLE Handler and the Web GUI Handler.

Web GUI Handler

Role

Implement an HTTP server to which an HTTP client can connect.

Serve as a transparent interface between the event generated in the GUI (i.e. the Web Client) and the Sensor Server. This means that when the implementation of the interface needs to be adjusted (e.g. from Web Client to a local native GUI), only this component needs to be replaced.

Context

The context of the Web GUI Handler is given in figure 5 and is formed by the Sensor Server and the Web Client.

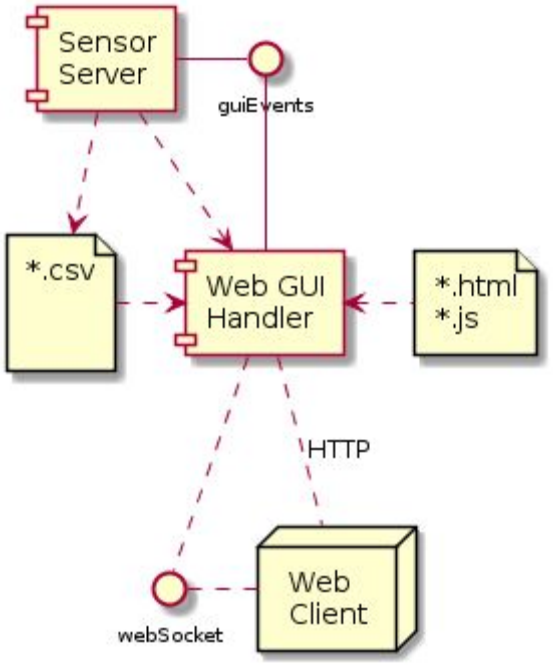


Figure 5: Content of the Web GUI Handler.

Interfacing

The Web GUI Handler interfaces with the filesystem to be able offer the recordings for download to the Web Client.

Furthermore, the Web GUI Handler functions as HTTP server for a Web Client to be able to connect and receive the HTML file to render.

To implement the interaction, the Web GUI Handler also offers a web socket where it takes the role of server for a client to connect.

The events that are handled by the **`eventHandler`** callback function specified as the argument of the constructor are simply the ones that are forwarded as received from the Web Client.

The interface functions of the Web GUI Handler are given in table 11.

Table 11: Web GUI Handler interface functions.

Method	Arguments	Description
`constructor`	• `eventHandler`	Constructs a new Web GUI Handler. The `eventHandler` argument is a callback function that points to a function in the Sensor Server to have relevant events handled by the Sensor Server.
`sendGuiEvent`	• `name` • `parameters`	Sends a GUI event with the specified name and parameters.

Web Client

Role

The Web Client will connect to the Web GUI Handler using HTTP and send a GET request to retrieve the HTML-file to render.

The HTML-file will include some JavaScript files to implement the client side scripting which are downloaded (i.e. 'fetched') when loading the HTML-file.

Context

The context of the Web Client is given in figure 6.

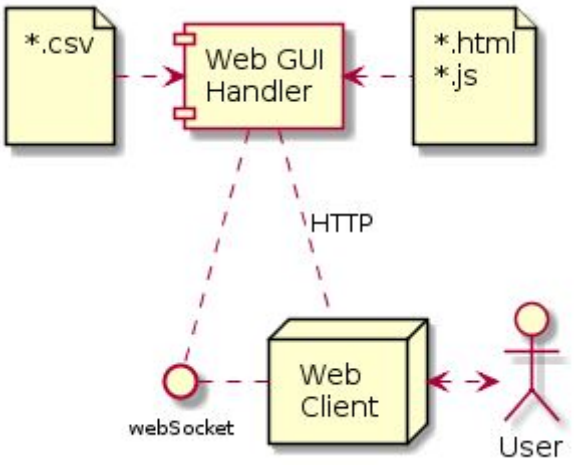


Figure 6: Context of the Web Client.

Interfacing

Part of the client side scripting is opening the client side of a web socket for which the Web GUI Handler implements the server side.

The Web Client can now emit events into this web socket which will be handled by the Web GUI Handler, which will forward them to the Sensor Server. Via this interface it is also possible to receive events from the Web GUI Handler that originated at the Sensor Server.

The events that can be generated are listed in table 12.

Table 12: Events generated by the GUI.

Event	Parameters	Description
`startScanning`	• -	Request to start scanning for new sensors.
`stopScanning`	• -	Stop scanning for new sensors.
`connectSensors`	• `addresses[]`	Try to establish a connection to the indicated sensors.
`stopConnectingSensors`	• -	Stop trying to connect the remaining sensors.
`disconnectSensors`	• -	Disconnect all the connected sensors.
`startMeasuring`	• `addresses[]`	Have the indicated sensors start their measurements.
`stopMeasuring`	• `addresses[]`	Disable the measurements on the indicated sensors.
`startRecording`	• `filename`	Start recording the sensor data to a file with the indicated name.
`stopRecording`	• -	Stop recording the sensor data to file and close the file.

The events that can be received are listed in table 13.

Table 13: Events received by the GUI.

Event	Parameters	Description
`ready`	• -	The set-up of the server finished.
`scanningStarted`	• -	As a response to the “startScanning” event, the server indicates that scanning started.
`scanningStopped`	• -	The scanning for new sensors stopped.
`sensorDiscovered`	• `name` • `address`	A new sensor is discovered.
`sensorConnected`	• `address`	A selected sensor connected.
`allSensorsConnected`	• -	All selected sensors are connected.
`sensorDisconnected`	• `address`	A connected sensor disconnected.
`sensorEnabled`	• `address`	The sensor is measuring.
`allSensorsEnabled`	• -	All selected sensors are enabled.
`sensorDisabled`	• `address`	The sensor stopped measuring.
`allSensorsDisabled`	• -	All selected sensors are disabled.
`sensorOrientation`	• `timestamp` • `address` • `q_w` • `q_x` • `q_y` • `q_z`	New data from the sensor.
`recordingStarted`	• -	Recording was started.

Next to the GUI events, also other events can be sent over the web socket in order to get access to application files, i.e. the recordings.

These file access events are listed in table 14.

Table 14: Events to access files.

Event	Parameters	Description
`getFileList`	• `-`	Get the list of recordings.
`fileList`	• `files[]`	A list of recordings as present on the file system of the server.
`deleteFiles`	• `files[]`	Delete the recordings with the indicated filenames.

Appendix

To be able to run node without sudo run the following command:

```
$ sudo setcap cap_net_raw+eip $(eval readlink -f `which node`)
```

This grants the node binary cap_net_raw privileges, so it can start/stop BLE.

Note: The above command requires setcap to be installed. If this is not the case, it can be installed using the following:

```
$ sudo apt-get install libcap2-bin
```