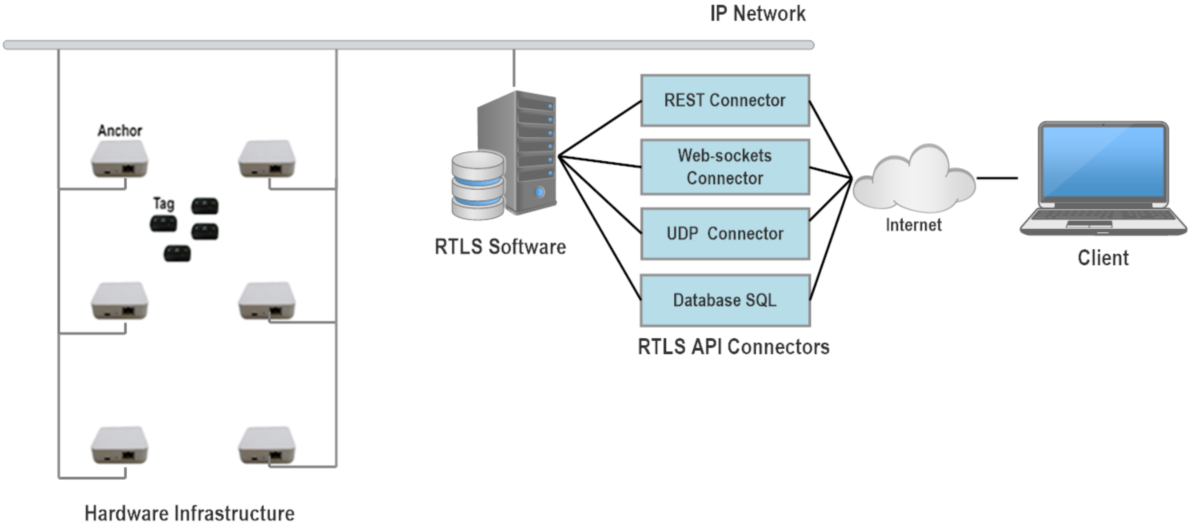
Application Note AN07
API Guide

API is used to give the user access to position data in Cartesian coordinates, inertial sensors' data, battery information, user button info and timestamps.

1 How to use the API

An overview of the communication architecture can be found in the figure below:



There are three different ways how to access the position data on server through the API. Each is more suitable for different application. See the infographics below:

Refresh rates:	300ms – 5s	100ms – 300ms	< 100ms
API access:	REST, Websockets	REST, Websockets	REST, UDP Stream Or Websockets

- REST can be used to access static and historic data, like information about Buildings, Floorplans, Tags, Anchors etc. It is the only connector that is able to get static data from the database.

- Websockets are used in applications with high refresh rates. **Websockets can only provide positioning data. To access static or historic data, please use the REST connector.**

- UDP Stream is used for extremely fast refresh rates. **UDP Stream can only provide positioning data. To access static or historic data, please use the REST connector.**

Each API Connector is more suitable for different types of data. The table below describes the feasibility of each connector to be used for a specific type of data:

Type of data	REST	Websockets	UDP Stream
Static data (Buildings, Floor plans etc.)	Yes	No	No
Historic data	Yes	No	No
Real-time positioning data	No	Yes	Yes
Real-time Inertial sensors data	No	Yes	Yes
Zones data	No	Yes	No
Change in Static data	Yes	Yes	No

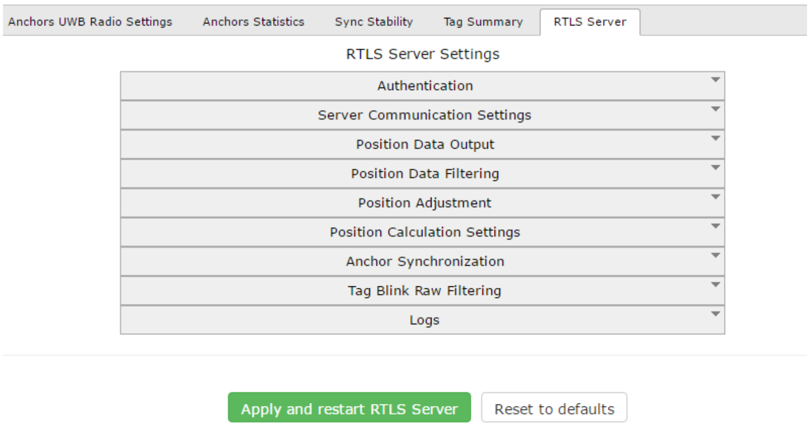
The data are stored on a host server in the form of a MySQL Database. The database can be accessed by the three options described above. The fourth option is to access the database directly, if one is very experienced with MySQL. **However, this is not recommended to gather position data in real-time and Sewio cannot provide any support here, since a user inexperienced in databases can damage the database storage.**

2 Real time data streaming and storing to database

The positioning data are streamed in real time into Sensmapserver, which visualizes the positions. There are multiple options on how to stream the data, while each is more suitable for different refresh rates. To change the data streaming into Sensmap, follow these steps:

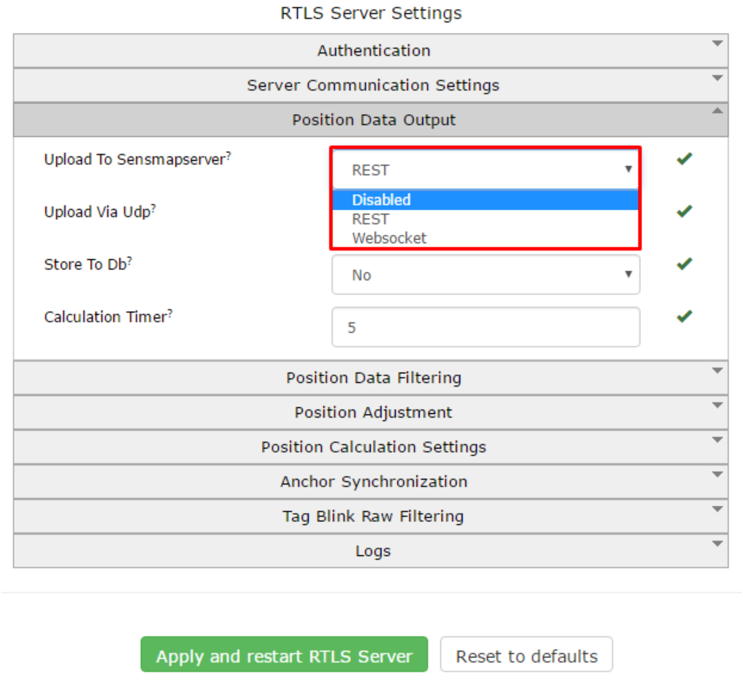
1) Navigate to the RTLS Manager by clicking its icon in RTLS Studio:

2) Go to the RTLS Server tab



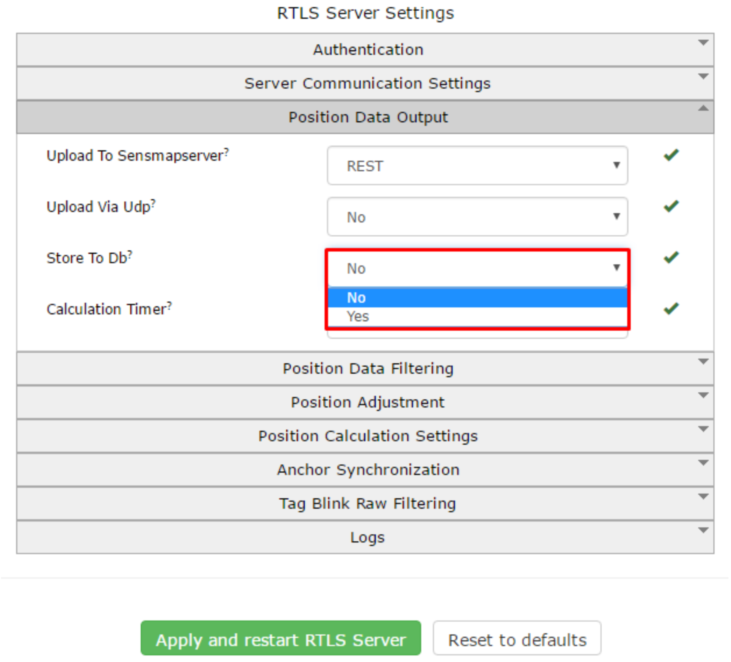
3) Click on Position Data Output

Here you will find the “Upload to Sensmapserver” parameter, which sets how the data will be uploaded into Sensmap. REST is the default setting, which is suitable for normal refresh rates. For higher refresh rates, it is better to switch to Websockets. If you use very high refresh rates (< 100ms), it is best to use the UDP Connector and set “Upload to Sensmapserver” to Disabled. **However this also means that positions won’t be visualized in Sensmap.**



Upload To Sensmapserver? REST ✓ (Disabled / REST / Websocket)
Upload Via Udp? ✓
Store To Db? No ✓
Calculation Timer? 5 ✓

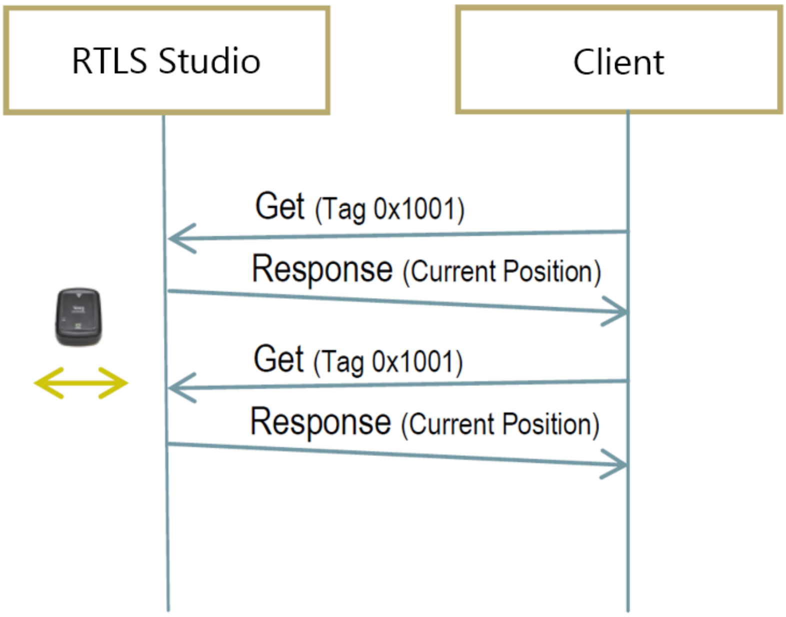
In order to store position data to database, you need to set the parameter “Store to Db” to Yes. If you set the parameter to No, positions won’t be stored into database and you won’t be able to access historic data.



Upload To Sensmapserver? REST ✓
Upload Via Udp? No ✓
Store To Db? No ✓ (No / Yes)
Calculation Timer? ✓

3 REST API

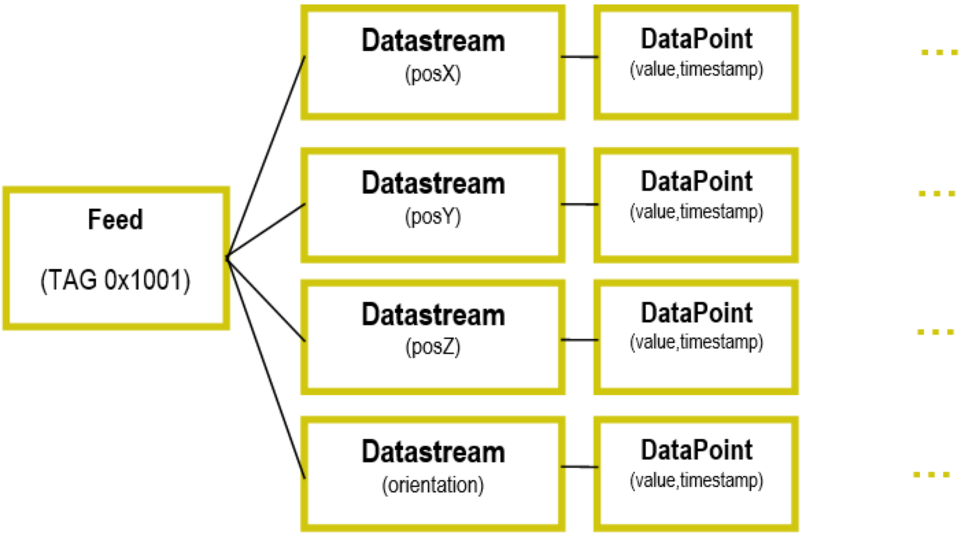
REST is a HTTP based interface. It consists of the following Methods: GET, CREATE, UPDATE, DELETE. All methods along with the responses are encapsulated in JSON format and transferred as HTTP requests. With each request, a response is given from the server. The communication is based on request-response architecture, which means that the user need to frequently request position updates to receive the positioning data, see the figure below:



3.1 Data Entities

Data entities have three categories:

- **Feed –** is a most general data representation of physical object or device such as Tag, Building, Zone Definition etc.
- **Datastream –** is a representation of physical phenomena such as position, battery level, orientation etc.
- **DataPoint –** is a representation of discrete measurement of datastream equipped with timestamp



There are other entities derived from basic Feed:

- **Anchor** - represents a physical Anchor device and its data
- **Tag** - represents a physical Tag device and its data
- **Building** - represents a physical Building and its data
- **Floorplan** - represents an image for a particular floor within a building
- **Zone** - represents a virtual Zone within a Building

3.2 When to use REST Interface

REST Interface is mainly useful for applications with very slow refresh rates (seconds) or to read static data from the database, like information about Feeds, namely:

- Tags
- Anchors,
- Buildings
- Floorplans

It can also be used to read historic position data. **REST interface should not be used to read fast real-time position data.** The user authentication is done by the use of an API Key.

3.3 Interactive REST Interface

You can try working with REST in our interactive API interface. To try it out, you have two options:

1) You can try it out at our online demo at the following website:

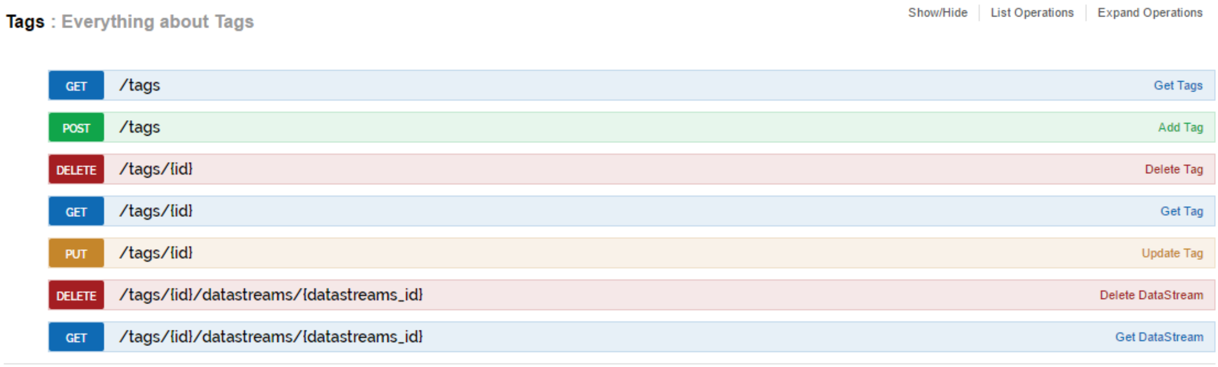
   http://www.rtlsstudio.com/studio/index.php/api-connectors-overview/

2) Or you can try it on your own version of RTLS Studio at the following address:

   IP_ADDRESS/studio/index.php/api-connectors-overview/

   Note: replace IP_ADDRESS, with the real IP address, the default is 192.168.225.2

An example of the Interactive REST API, where you can get information about Tags is in the figure below:



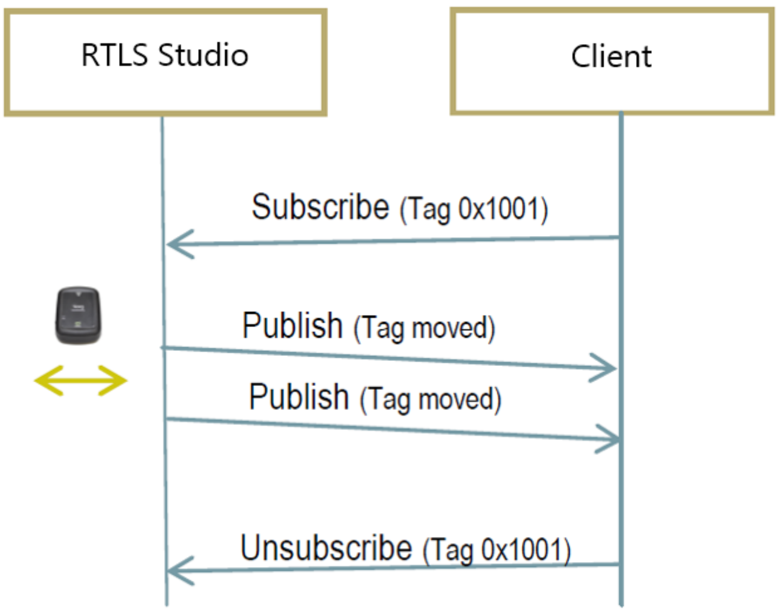
Tags : Everything about Tags

Show/Hide | List Operations | Expand Operations

GET	/tags	Get Tags
POST	/tags	Add Tag
DELETE	/tags/{id}	Delete Tag
GET	/tags/{id}	Get Tag
PUT	/tags/{id}	Update Tag
DELETE	/tags/{id}/datastreams/{datastreams_id}	Delete DataStream
GET	/tags/{id}/datastreams/{datastreams_id}	Get DataStream

4 Websockets API

Websockets is a protocol carried over single TCP connection heavily used in web based applications. This ensures the reliability of the data exchange. Communication is based on publish-subscribe model, where client can subscribe to a particular stream from Tag or Anchor and be immediately updated whenever its position is being changed. The interface is dedicated for real-time and read-only data exchange. See the communication architecture in the figure below:



4.1 When to use Websockets Interface

Websockets are ideal to read real-time position data. They are very lightweight and easy to use and are suitable for refresh rates up to 300 ms. They provide information about positioning data only. Rest of the data must be gathered via REST Connector. The user authentication is done by the use of an API Key.

4.2 Websockets Example

Now we well demonstrate how Webscokets can be utilized to gather position data from a Tag. There are three methods available:

- **Subscirbe:**

```
{"headers":{"X-ApiKey":"YOUR_KEY"},"method":"subscribe","resource":"/feeds/YOUR_FEED"}
```

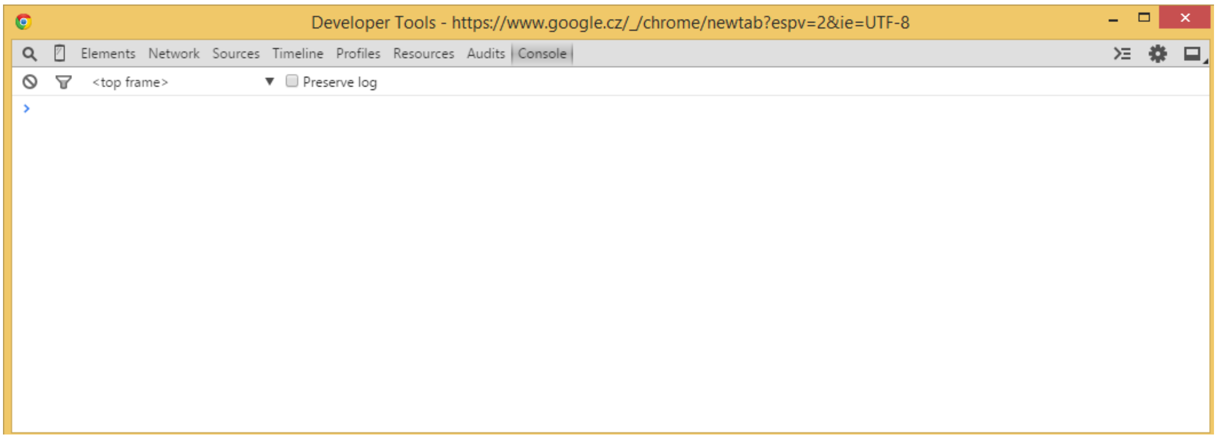
- **Unsubscribe**

```
{"headers":{"X-ApiKey":"YOUR_KEY"},"method":"unsubscribe","resource":"/feeds/YOUR_FEED"}
```

- **Put**

```
{"headers":{"X-ApiKey":"YOUR_KEY"},"method":"put",options:{savetoDB:false},
"body":{"id":FEED_ID,"datastreams":[{"id":"posX","current_value":VALUE},
{"id":"posY","current_value":VALUE}]}, "resource":"/feeds/FEED_ID"}
```

Let’s use the Developer Console from Google Chrome web browser to be an independent on programming. The console can be found in Menu -> More tools -> JavaScript Console (CRTL+SHIFT+J):

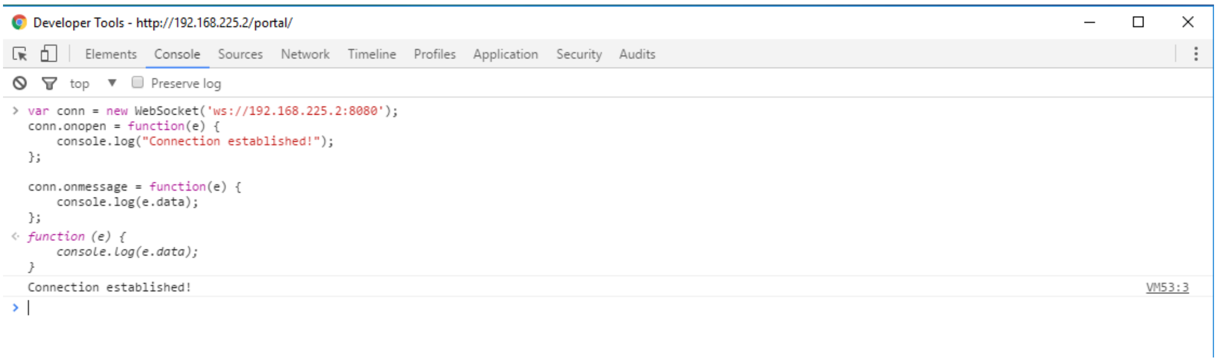


Firstly, Client must connect to Sensmap Server Websocket interface which resides implicitly on IP 195.113.243.99 and port 8080.

```
Var conn = new WebSocket('ws://195.113.243.99:8080');
conn.onopen = function(e) {
console.log("Connection established!");
};

conn.onmessage = function(e) {
console.log(e.data);
};
```

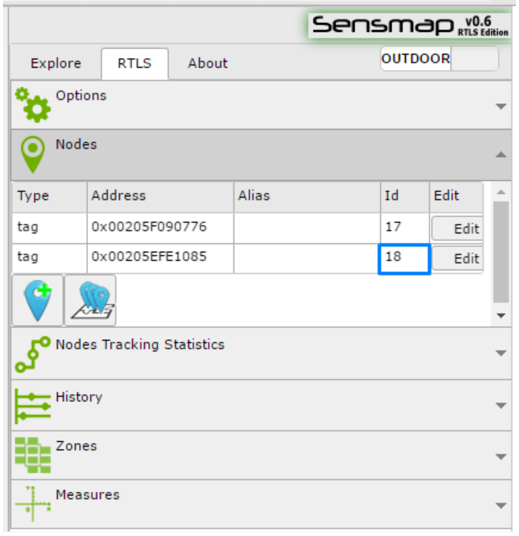
Note: You can use our demo's IP address or you can use the address of your RTLS Studio (default is 192.168.225.2).



```
> var conn = new WebSocket('ws://192.168.225.2:8080');
  conn.onopen = function(e) {
      console.log("Connection established!");
  };

  conn.onmessage = function(e) {
      console.log(e.data);
  };
<· function (e) {
      console.log(e.data);
  }
  Connection established!                                          VM53:3
> |
```

"Connection established!" should appear within the console window. Now we can subscribe to the Tag 0x00205EFE1085. Before that we need to find unique ID called FEED_ID. This can be done visually through Sensmap Visualization or via REST command GET FEEDs request.

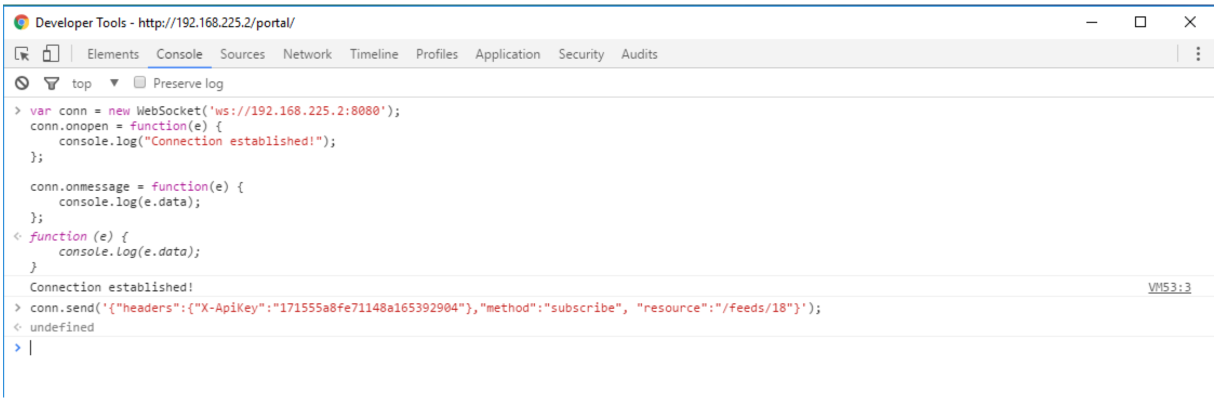


Type	Address	Alias	Id	Edit
tag	0x00205F090776		17	Edit
tag	0x00205EFE1085		18	Edit

From picture above we see that Tag 0x00205EFE1085 has assigned ID 18.

Now we can subscribe to its datastream with a default read-only key 171555a8fe71148a165392904:

```
conn.send('{"headers":{"X-ApiKey":"171555a8fe71148a165392904"},"method":"subscribe",
"resource":"/feeds/18"}');
```

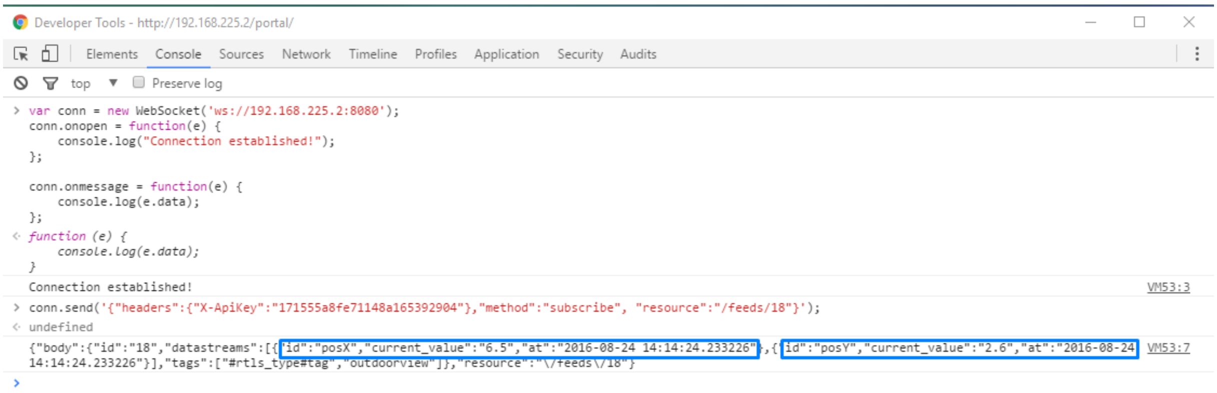


```
> var conn = new WebSocket('ws://192.168.225.2:8080');
  conn.onopen = function(e) {
      console.log("Connection established!");
  };

  conn.onmessage = function(e) {
      console.log(e.data);
  };
<· function (e) {
      console.log(e.data);
  }
  Connection established!                                                    VM53:3
> conn.send('{"headers":{"X-ApiKey":"171555a8fe71148a165392904"},"method":"subscribe", "resource":"/feeds/18"}');
<· undefined
> |
```

Then we need to update position of the Tag, therefore if the RTLS System is running, you may start moving the Tag.

Immediately after the position is changed updated information should appear in console output. In our example position was updated to values [6.5, 2.6], see marked variables.



```
> var conn = new WebSocket('ws://192.168.225.2:8080');
  conn.onopen = function(e) {
      console.log("Connection established!");
  };

  conn.onmessage = function(e) {
      console.log(e.data);
  };
<· function (e) {
      console.log(e.data);
  }
  Connection established!                                                    VM53:3
> conn.send('{"headers":{"X-ApiKey":"171555a8fe71148a165392904"},"method":"subscribe", "resource":"/feeds/18"}');
<· undefined
  {"body":{"id":"18","datastreams":[{"id":"posX","current_value":"6.5","at":"2016-08-24 14:14:24.233226"},{"id":"posY","current_value":"2.6","at":"2016-08-24   VM53:7
  14:14:24.233226"}],"tags":["#rtls_type#tag","outdoorview"]},"resource":"\/feeds\/18"}
>
```

5 UDP Stream API

UDP Connector is dedicated for Tags' positions upload for very high speed, with minimum latency where other connectors are too slow.

UDP was designed with simplicity in mind, so there is no need to subscribe, no need for polling, etc. If there is some new position for any Tag, this position will be sent via UDP to designated IP:port. However this also means that unlike TCP the connection is not reliable and there can be some lost packets.

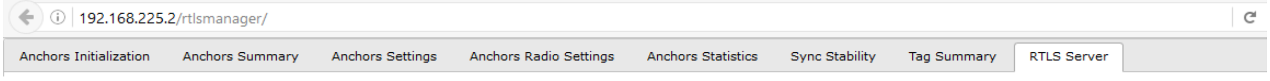
Example of a UDP packet:

```
{"id":"18","datastreams":[{"id":"posX","current_value":"6.5","at":"2016-08-24
14:14:24.233226"},{"id":"posY","current_value":"2.6","at":"2016-08-24
14:14:24.233226"}]}
```

5.1 How to use the UDP Stream API

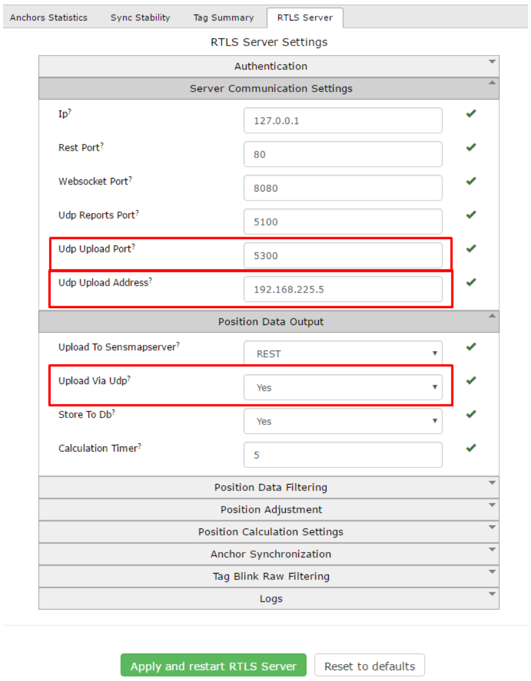
First the UDP Upload must be enabled in RTLS Studio.

1) Access “RTLS Server” tab in RTLS Manager:



192.168.225.2/rtlsmanager/

2) There are three settings which can be configured: UDP Upload Port, UDP Upload Address and most importantly, Upload via UDP. Calculated positions will be sent to UDP Upload Address:UDP Upload Port, if Upload Via UDP is set to **Yes**. After setting these three values press **Apply and restart RTLS Server.**



RTLS Server Settings

Server Communication Settings

Ip?	127.0.0.1
Rest Port?	80
Websocket Port?	8080
Udp Reports Port?	5100
Udp Upload Port?	5300
Udp Upload Address?	192.168.225.5

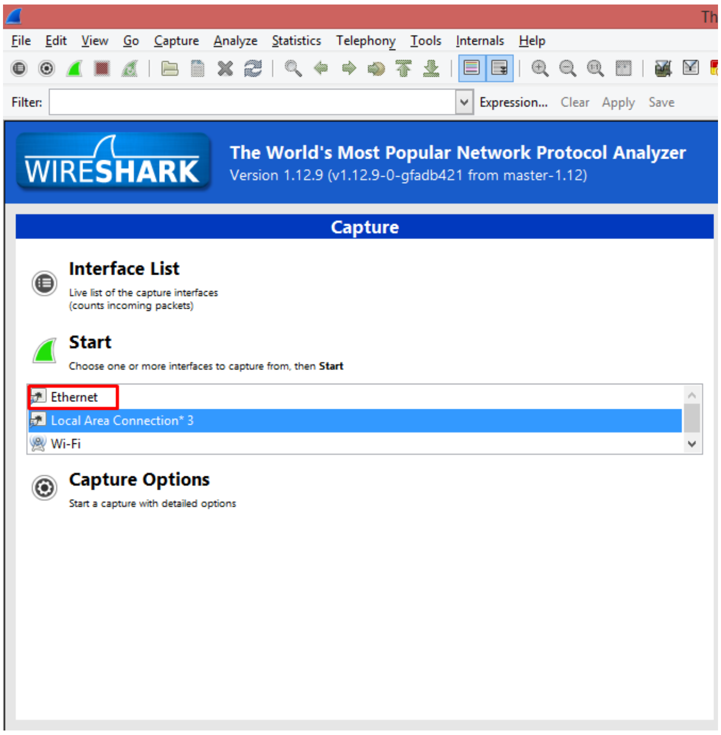
Position Data Output

Upload To Sensmapserver?	REST
Upload Via Udp?	Yes
Store To Db?	Yes
Calculation Timer?	5

These steps enable the UDP Stream on selected IP address and port. Now we need an application, that will listen on that socket to capture the positioning data. Here we will demonstrate how these data can be captured by Wireshark.

1) Open Wireshark.

2) Select your Ethernet interface and click “Start”


Th
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
WIRESHARK

The World's Most Popular Network Protocol Analyzer
Version 1.12.9 (v1.12.9-0-gfadb421 from master-1.12)

Capture

Interface List
Live list of the capture interfaces (counts incoming packets)

Start
Choose one or more interfaces to capture from, then **Start**

- Ethernet
- Local Area Connection* 3
- Wi-Fi

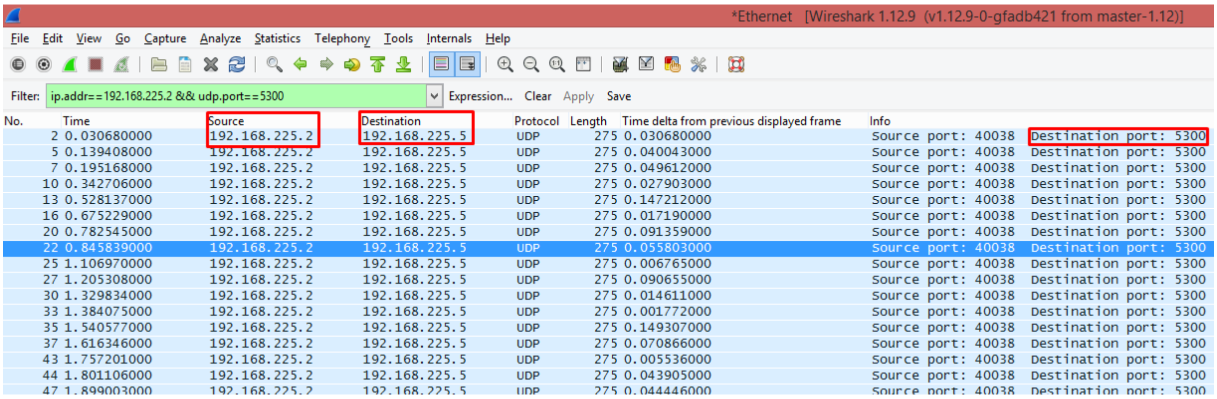
Capture Options
Start a capture with detailed options

3) Then you should see a stream of data. To filter out only the desired data, enter the source IP address and UDP port in the “Filter:” window by using the following command:

ip.addr==IP_ADDRESS && udp.port==UDP_PORT

(Default: IP_ADDRESS = 192.168.225.2, UDP_PORT = 5300)

4) Then you should see a stream of positioning data from the server, as shown in the figure below. You can see the source address, destination port (which you set earlier) and destination address. In this example, the address of the host computer is 192.168.225.5.

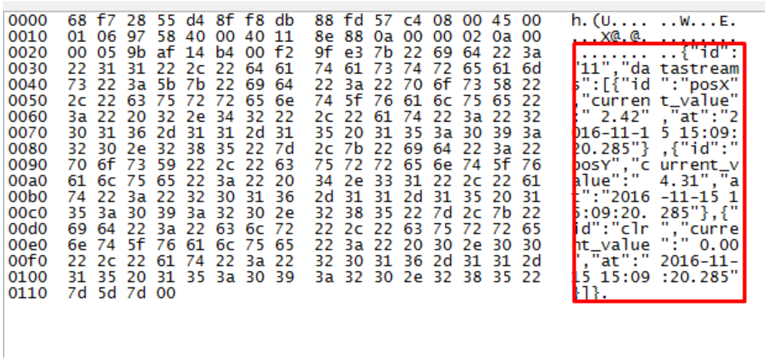

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

*Ethernet [Wireshark 1.12.9 (v1.12.9-0-gfadb421 from master-1.12)]

Filter: ip.addr==192.168.225.2 && udp.port==5300

No.	Time	Source	Destination	Protocol	Length	Time delta from previous displayed frame	Info
2	0.030680000	192.168.225.2	192.168.225.5	UDP	275	0.030680000	Source port: 40038 Destination port: 5300
5	0.139408000	192.168.225.2	192.168.225.5	UDP	275	0.040043000	Source port: 40038 Destination port: 5300
7	0.195168000	192.168.225.2	192.168.225.5	UDP	275	0.049612000	Source port: 40038 Destination port: 5300
10	0.342706000	192.168.225.2	192.168.225.5	UDP	275	0.027903000	Source port: 40038 Destination port: 5300
13	0.528137000	192.168.225.2	192.168.225.5	UDP	275	0.147212000	Source port: 40038 Destination port: 5300
16	0.675229000	192.168.225.2	192.168.225.5	UDP	275	0.017190000	Source port: 40038 Destination port: 5300
20	0.782545000	192.168.225.2	192.168.225.5	UDP	275	0.091359000	Source port: 40038 Destination port: 5300
22	0.845839000	192.168.225.2	192.168.225.5	UDP	275	0.055803000	Source port: 40038 Destination port: 5300
25	1.106970000	192.168.225.2	192.168.225.5	UDP	275	0.006765000	Source port: 40038 Destination port: 5300
27	1.205308000	192.168.225.2	192.168.225.5	UDP	275	0.090655000	Source port: 40038 Destination port: 5300
30	1.329834000	192.168.225.2	192.168.225.5	UDP	275	0.014611000	Source port: 40038 Destination port: 5300
33	1.384075000	192.168.225.2	192.168.225.5	UDP	275	0.001772000	Source port: 40038 Destination port: 5300
35	1.540577000	192.168.225.2	192.168.225.5	UDP	275	0.149307000	Source port: 40038 Destination port: 5300
37	1.616346000	192.168.225.2	192.168.225.5	UDP	275	0.070866000	Source port: 40038 Destination port: 5300
43	1.757201000	192.168.225.2	192.168.225.5	UDP	275	0.005536000	Source port: 40038 Destination port: 5300
44	1.801106000	192.168.225.2	192.168.225.5	UDP	275	0.043905000	Source port: 40038 Destination port: 5300
47	1.899003000	192.168.225.2	192.168.225.5	UDP	275	0.044446000	Source port: 40038 Destination port: 5300

5) Then if you click on some packet in the stream, you can see the decoded data in the window at the bottom:



```
0000   68 f7 28 55 d4 8f f8 db  88 fd 57 c4 08 00 45 00   h.(U.... ..W...E.
0010   01 06 97 58 40 00 40 11  8e 88 0a 00 00 02 0a 00   ...X@.@. ........
0020   00 05 9b af 14 b4 00 f2  9f e3 7b 22 69 64 22 3a   ........ ..{"id":
0030   22 31 31 22 2c 22 64 61  74 61 73 74 72 65 61 6d   "11","da tastream
0040   73 22 3a 5b 7b 22 69 64  22 3a 22 70 6f 73 58 22   s":[{"id ":"posX"
0050   2c 22 63 75 72 72 65 6e  74 5f 76 61 6c 75 65 22   ,"curren t_value"
0060   3a 22 20 32 2e 34 32 22  2c 22 61 74 22 3a 22 32   :" 2.42" ,"at":"2
0070   30 31 36 2d 31 31 2d 31  35 20 31 35 3a 30 39 3a   016-11-1 5 15:09:
0080   32 30 2e 32 38 35 22 7d  2c 7b 22 69 64 22 3a 22   20.285"} ,{"id":"
0090   70 6f 73 59 22 2c 22 63  75 72 72 65 6e 74 5f 76   posY","c urrent_v
00a0   61 6c 75 65 22 3a 22 20  34 2e 33 31 22 2c 22 61   alue":"  4.31","a
00b0   74 22 3a 22 32 30 31 36  2d 31 31 2d 31 35 20 31   t":"2016 -11-15 1
00c0   35 3a 30 39 3a 32 30 2e  32 38 35 22 7d 2c 7b 22   5:09:20. 285"},{"
00d0   69 64 22 3a 22 63 6c 72  22 2c 22 63 75 72 72 65   id":"clr ","curre
00e0   6e 74 5f 76 61 6c 75 65  22 3a 22 20 30 2e 30 30   nt_value ":" 0.00
00f0   22 2c 22 61 74 22 3a 22  32 30 31 36 2d 31 31 2d   ","at":" 2016-11-
0100   31 35 20 31 35 3a 30 39  3a 32 30 2e 32 38 35 22   15 15:09 :20.285"
0110   7d 5d 7d 00                                        }]}.
```

You can see the data in JSON format, just like with other API connectors.