1SYSTEM®

bf1systems Advertisement API Developer guide

from

bf1systems

1system@bf1systems.com

Version 1.9.0 Revision 5

Contents

Introduction

This document covers the functionality of the bf1systems Advertisement API, including guidance on usage with examples written in Python, Javascript and C#. The API provides real-time sensor data from multiple DCSs through an SSE stream. The API allows for multiple clients to connect simultaneously. The API can also get and set tag information associated with sensor addresses. Resources stored in the database are shared by all clients connecting to the API. Since the API stores tags in a local database, tags persist across restarts and updates.

Agreement

> **ⓘ Note**
> If this document has been printed then it cannot be guaranteed that it is the most recent version. Please contact bf1systems for the latest release.

How does the API system work?

Components

DCS

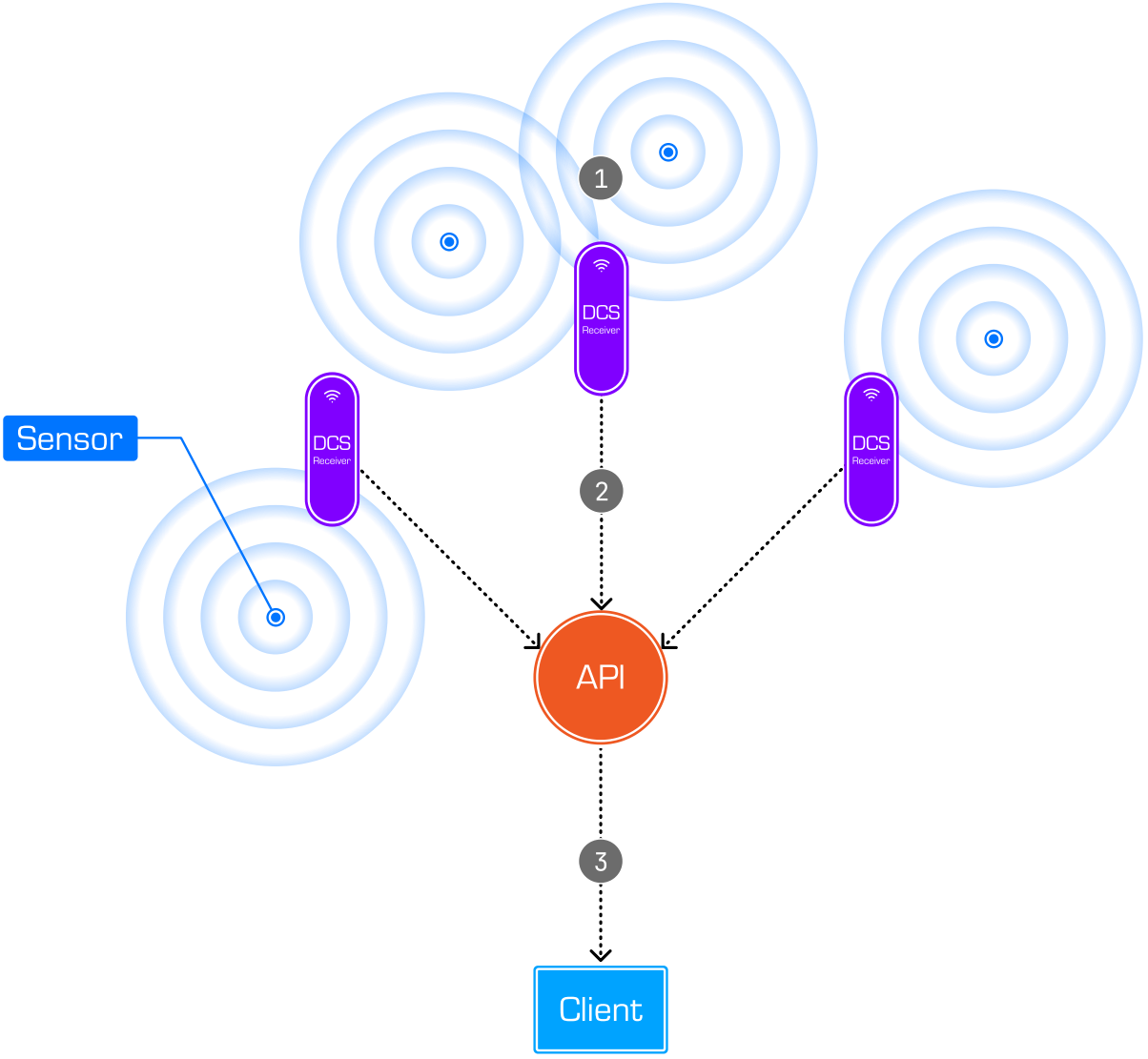
The DCS or device communication service handles over-the-air wireless connections with the sensors. The DCS transfers this information via a TCP/IP connection to the API. Multiple DCSs can be configured to connect to the API. The port number for this connection is configurable, see the DCS installer guide for more details.

API

The API or application programming interface processes data from the configured DCSs and handles connections and requests from clients.

Client

The client machine connects to the API via a configurable endpoint (see API installation guide for more details). Multiple clients can access the API simultaneously, all sharing the same resources (eg. tags).



Stage 1

1. Sensors transmit data.
2. Data is received by DCSs (receivers).

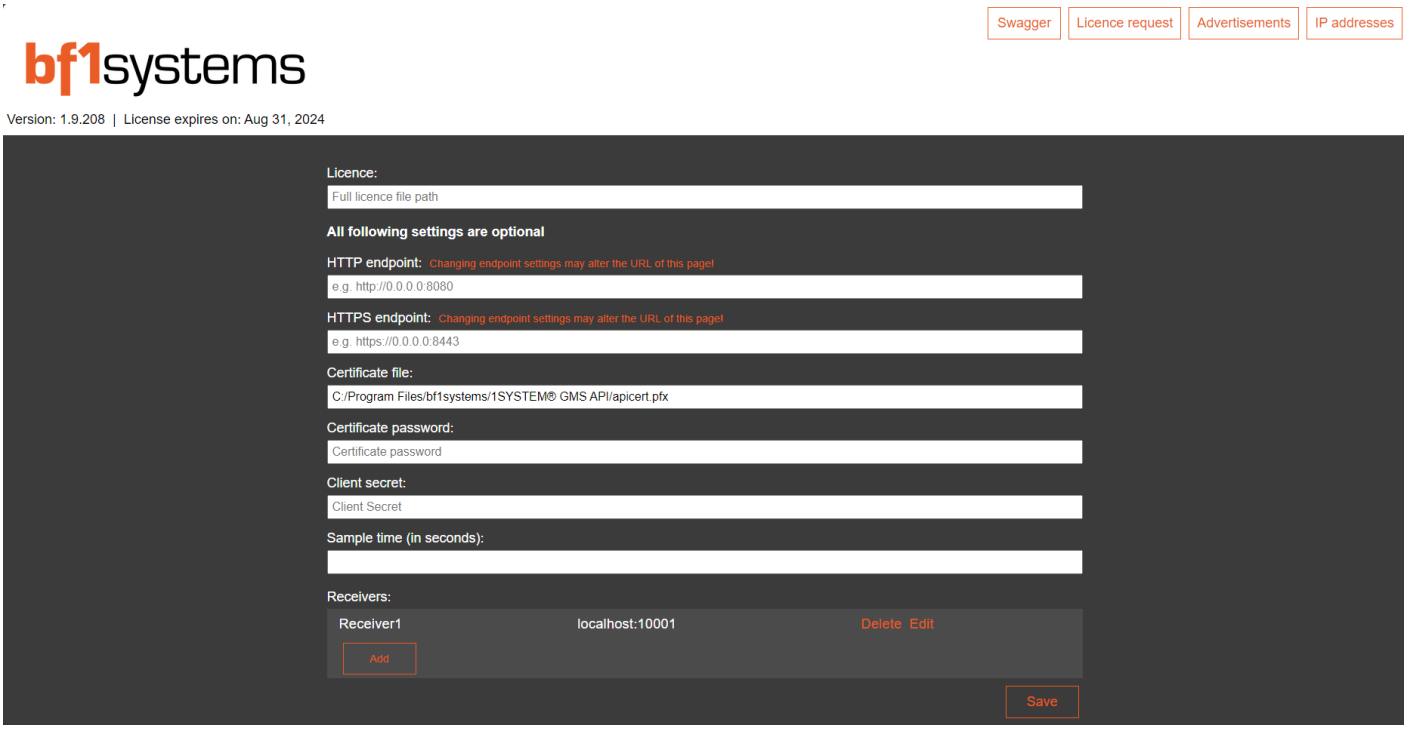
Stage 2

1. DCSs send the received data to the API.
2. The API combines data streams from all DSCs and decrypts it.

Stage 3

1. The API sends the combined data stream to the client via an open SSE connection.

API Configuration Page



The API configuration page can be accessed via the `<base_url>` in a browser. All configuration fields are optional; if left unchanged, default settings will be used. *See table below for default values.*

The configuration page also provides quick access to the Swagger page, license request route, advertisements route and the current IP addresses of the machine running the API.

API Configuration Options

Configuration Item	Description	Default
License	Defines the local path to the API license file. A license is required to operate the API.	-
`http` endpoint	Defines a `base_url` for connecting to the API. Using `http` does not require the use of a certificate.	`http://0.0.0.0:8080`
`https` endpoint	Defines a `base_url` for **securely** connecting to the API. Using `https` **does** require the use of a certificate.	`https://0.0.0.0:8443`
Certificate	Defines the local path to the certificate file. The `certificate` is used to encrypt the connection when using the `https` protocol to connect to the API.	-
Certificate password	Password to use the `https` certificate.	-
Client Secret	The client secret is used for authorise access to the `/Advertisements` route. This acts as a password for accessing sensor data.	-
Sample time	The sample time defines an interval in which signals received by multiple receivers will be be batched into one advertisement.	-

ⓘ Note

A certificate for `https` is installed as part of the API installation process however, this certificate is self-signed and may not be trusted by the client. A certificate from a trusted provider is recommended for production environments.

⚠ Warning

When using the `http` protocol, data sent to/from the API is **not** encrypted. This means that all data sent and received, including the `clientSecret` in the request, may be visible to others on the network.

Certificates

A certificate is required in order to use `https` on the API's endpoint. This certificate can either be self-signed or issued by a trusted provider. If your certificate is self-signed, connections and requests to the API endpoint may be blocked by safety features built into your operating system. This issue arises from the client's inability to trust the certificate provider. In such case, it may be necessary to suppress warnings and errors caused by certificate trust issues when connecting to an endpoint. See the code examples below on how to bypass these messages.

Before you start

Before connecting to the API, ensure that you:

- Replace `<base_url>` with the actual base URL of the API in the examples provided in this document.
- Use the correct connection protocol in your base URL, either `http` or `https`, depending on the API configuration.
- Request parameters marked with an asterisk (*) are required.

Microsoft Excel Add-in

To view the sensor data using Microsoft Excel, you can use the bf1systems Excel Add-in. The add-in allows you to connect to the API and view sensor data in real-time.

Installation

Step 1

Open Microsoft Excel and navigate to `File > Options > Add-ins`.

Step 2

At the bottom of the window, select `Excel Add-ins` from the `Manage` dropdown and click `Go`.

Step 3

Click `Browse` and navigate to the location of the Add-In file (`bf1s.AdvertisementApiExcelAddIn-AddIn64.xll`) and click `Open`.

Step 4

Ensure the add-in is checked and click `OK`.

Step 5

Open the sample document to start viewing the sensor data. Go to `File > Open` and navigate to the location of the sample document (`Excel PoC.xlsm`) and click `Open`.

Step 6

Ensure the Add-In is enabled by clicking `Enable Content`.

Step 7

Ensure the API address and secret are correct in cells `B2` and `B3` respectively.

Step 8

Click `Start` to begin receiving sensor data.

Basic use

The Excel Add-In uses pre-defined functions to fetch sensor data based on an index. For example, the function `=GetSensorAddress(0)` will return the sensor address at index `0` (first sensor in the list). The function `=GetIndexForMacAddress` will return the index of a sensor given its MAC address. Using this function on a sensor that has not yet been received will return `#N/A` until the sensor is received. To find a full list of functions, click the `Insert function` button in the Excel ribbon. Select the `bf1s.AdvertisementAPIExcelAddIn` category to view all available functions.

Authentication

If a client secret is set when configuring the API, replace `<client_secret>` with your client secret, otherwise,this field/parameter can be omitted. This is required for protected routes such as `/Advertisements`, however if no client secret has been set then `ClientSecret` can be omitted from the request. `ClientSecret` can be included in the request as either a query parameter, or in the request header (see examples below).

Query Parameter

```
https://<base_url>/Advertisements?ClientSecret=<client_secret>
```

Request Header

```
{
  "ClientSecret": "<client_secret>"
}
```

Advertisements Route

`GET` Advertisements

Connects to a feed of sensor advertisements.

Requests

Method	Path	Headers	Query
`GET`	`/Advertisements`	`ClientSecret`	`ClientSecret`

Responses:

Code	Description	Body	Content Types
`200`	`Success`	Advertisement	`text/plain` `application/json` `text/json`
`404`	`Not found`	`Client secret does not match!`	-
`401`	`Unauthorized`	`An error occurred while parsing the license...`	-

Python Example

```python
import sseclient

# Specify the URL
url = '<base_url>/Advertisements'

headers = {"ClientSecret": "<client_secret>"}

# establish sse connection, suppressing self-signed certificate verification
advertisements = sseclient.SSEClient(url,headers=headers,verify=False)
for advertisement in advertisements:
    # Print each advertisement
    print(advertisement)
```

Javascript Example

```
var EventSource = require('eventsource');

const url = '<base_url>/Advertisements?ClientSecret=<client_secret>';

// Create event source, suppressing unauthorised (self-signed) certificate error
var source = new EventSource(url, { https: { rejectUnauthorized: false } });

// Log advertisements to console
source.onmessage = function (message) {
  console.log(message.data);
};
```

C# Example

```
// Create a HttpClientHandler
var handler = new HttpClientHandler();

// Set the ServerCertificateCustomValidationCallback to allow self-signed certificates
handler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) =>
{
    return true;
};

// Create the http client
HttpClient client = new HttpClient(handler);

// Specify the URL
string url = "<base_url>/Advertisements?ClientSecret=<client_secret>";

// Read the data stream
using (var streamReader = new StreamReader(await client.GetStreamAsync(url)))
{
    while (!streamReader.EndOfStream)
    {
        var advertisement = await streamReader.ReadLineAsync();

        // Write each advertisement to console
        Console.WriteLine(advertisement);
    }
}
```

License Request Route

`GET` License Request

A license request is required to obtain a license to use the API. The license request route enables you to generate a license request file (`.bf1licreq`).

> **ⓘ Note**
> A license request file is **machine specific**. The license request route must be called from the **same machine** that is running the API (i.e `https://localhost:<port_number>/LicenseRequest`).

Requests

Method	Path
`GET`	`/LicenseRequest`

Responses

Code	Description	Body	Content Types
`200`	`Success`	License Request	`application/bf1licreq`

Python Example

```
import requests

# Specify the URL
url = '<base_url>/LicenseRequest'

# Send the GET request
response = requests.get(url,verify=False)

# Print the response
print(response.text)
```

Javascript Example

```
const request = require('request');

// Specify the URL
const url = '<base_url>/LicenseRequest';

// Send the GET request
request.get(
  url,
  {
    rejectUnauthorized: false,
  },
  (err, res, body) => {
    // Log the response to console
    console.log(body);
  }
);
```

C# Example

```
// Create an HttpClientHandler instance
var handler = new HttpClientHandler();

// Set the ServerCertificateCustomValidationCallback to allow self-signed certificates
handler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) =>
{
    return true;
};

// Create an HttpClient instance
var client = new HttpClient(handler);


// Set the base address and headers
client.BaseAddress = new Uri("<base_url>");

// Make a GET request
var response = await client.GetAsync("/LicenseRequest");

// Read the response as a string
var responseString = await response.Content.ReadAsStringAsync();

// Write the response to console
Console.WriteLine(responseString);
```

Sensors Route

`GET` Sensors

Retrieve an array of sensor addresses that have tags attached to them.

Requests

Method	Path
`GET`	`/Sensors`

Responses

Code	Description	Body	Content Types

`200`	`Success`	Sensor Array	`text/plain`
			`application/json`
			`text/json`

Python Example

```
import requests

# Specify the URL
url = '<base_url>/Sensors'

# Send the GET request
response = requests.get(url,verify=False)

# Print the response
print(response.text)
```

Javascript Example

```
const request = require('request');

// Specify the URL
const url = '<base_url>/Sensors';

// Send the GET request
request.get(
  url,
  {
    rejectUnauthorized: false,
  },
  (err, res, body) => {
    // Log the response to console
    console.log(body);
  }
);
```

C# Example

```
// Create an HttpClientHandler instance
var handler = new HttpClientHandler();

// Set the ServerCertificateCustomValidationCallback to allow self-signed certificates
handler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) =>
{
    return true;
};

// Create an HttpClient instance
var client = new HttpClient(handler);


// Set the base address and headers
client.BaseAddress = new Uri("<base_url>");

// Make a GET request
var response = await client.GetAsync("/LicenseRequest");

// Read the response as a string
var responseString = await response.Content.ReadAsStringAsync();

// Write the response to console
Console.WriteLine(responseString);
```

Tags Route

`GET` Tags (by Sensor Address)

Retrieve an array of tag metadata associated with a specific sensor address.

Requests

Method	Path	Parameters
`GET`	`/<sensor_address>`	`sensorAddress` *

Responses

Code	Description	Body	Content Types
`200`	`Success`	Tag Info Array	`text/plain` `application/json` `text/json`
`404`	`Not Found`	-	-

Python Example

```
import requests

# Specify the URL
url = '<base_url>/Tags/<sensor_address>'

# Send the GET request
response = requests.get(url,verify=False)

# Print the response
print(response.text)
```

Javascript Example

```
const request = require('request');

// Specify the URL
const url = '<base_url>/Tags/<sensor_address>';

// Send the GET request
request.get(
  url,
  {
    rejectUnauthorized: false,
  },
  (err, res, body) => {
    // Log the response to console
    console.log(body);
  }
);
```

C# Example

```
// Create an HttpClientHandler instance
var handler = new HttpClientHandler();

// Set the ServerCertificateCustomValidationCallback to allow self-signed certificates
handler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) =>
{
    return true;
};

// Create an HttpClient instance
var client = new HttpClient(handler);

// Set the base address and headers
client.BaseAddress = new Uri("<base_url>");

// Make a GET request
var response = await client.GetAsync("/Tags/<sensor_address>");

// Read the response as a string
var responseString = await response.Content.ReadAsStringAsync();

// Write the response to console
Console.WriteLine(responseString);
```

`GET` Tag (by Sensor Address and Key)

Retrieve a specific tag value associated with a sensor address.

Requests

Method	Path	Parameters
`GET`	`/Tags/<sensor_address>/<tag_key>`	`sensorAddress` * `tagKey` *

Responses

Code	Description	Body	Content Types

`200`	`Success`	Tag (JSON)	`text/plain`
			 `application/json`
			 `text/json`

`404`	`Not Found`	-	-

Python Example

```python
import requests

# Specify the URL
url = '<base_url>/Tags/<sensor_address>/<tag_key>'

# Send the GET request
response = requests.get(url,verify=False)

# Print the response
print(response.text)
```

Javascript Example

```
const request = require('request');

// Specify the URL
const url = '<base_url>/Tags/<sensor_address>/<tag_key>';

// Send the GET request
request.get(
  url,
  {
    rejectUnauthorized: false,
  },
  (err, res, body) => {
    // Log the response to console
    console.log(body);
  }
);
```

C# Example

```
// Create an HttpClientHandler instance
var handler = new HttpClientHandler();

// Set the ServerCertificateCustomValidationCallback to allow self-signed certificates
handler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) =>
{
    return true;
};

// Create an HttpClient instance
var client = new HttpClient(handler);


// Set the base address and headers
client.BaseAddress = new Uri("<base_url>");

// Make a GET request
var response = await client.GetAsync("/Tags/<sensor_address>/<tag_key>");

// Read the response as a string
var responseString = await response.Content.ReadAsStringAsync();

// Write the response to console
Console.WriteLine(responseString);
```

`POST` Tag

Create or update a tag value assigned to a sensor address.

Requests

Method	Path	Parameters	Body	Content Types
`POST`	`/Tags/<sensor_address>/<tag_key>`	`sensorAddress` * `tagKey` *	Tag (JSON)	`text/plain` `application/octet-stream`

Responses

Code	Description	Body	Content Types
`200`	`Success`	Tag Info	`text/plain` `application/json` `text/json`

Python Example

```
import requests

# Specify the URL
url = '<base_url>/Tags/<sensor_address>/<tag_key>'

# Specify the headers
headers = {"Content-Type": "text/plain"}

# Specify the data
data = {"ExampleProperty": "Example Value"}

# Send the POST request
response = requests.post(url,headers=headers,json=data,verify=False)

# Print the response
print(response.text)
```

Javascript Example

```
const request = require('request');

// Specify the URL
const url = '<base_url>/Tags/<sensor_address>/<tag_key>';

// Specify the data
const data = { ExampleProperty: 'Example Value' };

// Send the POST request
request.post(
  url,
  {
    json: data,
    rejectUnauthorized: false,
    headers: { 'Content-Type': 'text/plain' },
  },
  (err, res, body) => {
    // Log the response to console
    console.log(body);
  }
);
```

C# Example

```
// Create an HttpClientHandler instance
var handler = new HttpClientHandler();

// Set the ServerCertificateCustomValidationCallback to allow self-signed certificates
handler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) =>
{
    return true;
};

// Create an HttpClient instance
var client = new HttpClient(handler);

// Add media type headers to request
client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue("text/plain"));

// Set the base address and headers
client.BaseAddress = new Uri("<base_url>");

// Create an HttpContent object with the data to send
var content = new StringContent("{\"ExampleProperty\":\"Example Value\"}",
             Encoding.UTF8, new MediaTypeWithQualityHeaderValue("text/plain"));

// Make a POST request
var response = await client.PostAsync("/Tags/<sensor_address>/<tag_key>", content);

// Write the response to console
Console.WriteLine(response.StatusCode);
```

`DELETE` Tag

Delete a specific tag associated with a sensor address.

Requests

Method	Path	Parameters	Content Types
`DELETE`	`/Tags/<sensor_address>/<tag_key>`	`sensorAddress` * `tagKey` *	`text/plain` `application/octet-stream`

Responses

Code	Description
`200`	`Success`
`404`	`Not Found`

`DELETE` Tags

Delete all tags associated with a sensor address.

Requests

Method	Path	Parameters	Content Types
`DELETE`	`/Tags/<sensor_address>/`	`sensorAddress`	`text/plain` `application/octet-stream`

Responses

Code	Description
`200`	`Success`
`404`	`Not Found`

`DELETE` All Tags

Delete all tags in the database.

Requests

Method	Path	Parameters	Content Types
`DELETE`	`/Tags/`	-	`text/plain` `application/octet-stream`

Responses

Code	Description
`200`	`Success`
`404`	`Not Found`

> ⚠ **Warning**
> This action will delete **all tags** in the database. This action cannot be undone.

Python Example

```python
import requests

# Specify the URL
url = '<base_url>/Tags/<sensor_address>/<tag_key>'

# Send the GET request
response = requests.delete(url,verify=False)

# Print the response
print(response.text)
```

Javascript Example

```
const request = require('request');

// Specify the URL
const url = '<base_url>/Tags/<sensor_address>/<tag_key>';

// Send the DELETE request
request.delete(
  url,
  {
    rejectUnauthorized: false,
  },
  (err, res, body) => {
    // Log the response to console
    console.log(body);
  }
);
```

C# Example

```
// Create an HttpClientHandler instance
var handler = new HttpClientHandler();

// Set the ServerCertificateCustomValidationCallback to allow self-signed certificates
handler.ServerCertificateCustomValidationCallback = (sender, cert, chain, sslPolicyErrors) =>
{
    return true;
};

// Create an HttpClient instance
var client = new HttpClient(handler);

// Set the base address and headers
client.BaseAddress = new Uri("<base_url>");

// Make a DELETE request
var response = await client.DeleteAsync("/Tags/<sensor_address>/<tag_key>");

// Write the response to console
Console.WriteLine(response.StatusCode);
```

Data Schema

Advertisement

Schema


```
{
  "sensorAddress": "integer",
  "sensorName": "string",
  "sensorType": "string",
  "timestamp": "string",
  "receivers": "object",
  "sequence": "integer",
  "securityCode": "string",
  "tagsLastModifiedTimestamp": "integer",
  "errorFlags": "integer",
  "rbl": "float",
  "temperature": "float",
  "pressure": "float",
  "vehicleId": "integer",
  "wheelPosition": "integer",
  "tyreType": "integer",
  "tyreSet": "integer",
  "irCoreTemperature1": "float",
  "irCoreTemperature2": "float",
  "version": "string",
  "faultCode": "integer",
  "pixelCount": "integer",
  "pixelTemperatures": "object",
  "firmwareVersion": "string",
  "moving": "string"
}
```

Field	Format	Range	Units	Example
`sensorAddress` *MAC address encoded as decimal*	`Integer`	-	-	31013720310906
`sensorType`	`String`	IR / TPMS	-	IR
`timestamp` *Timestamp sensor was received*	`String`	-	-	2024-01-18T15:41:32…
`receivers` *Names and signal strengths of receivers detecting the sensor*	`object`	-	-	*see below
`sensorName`	`String`	-	-	1SIR4424
`sequence` *Sensor sequence number*	`Integer`	0 - 63	-	8
`securityCode`	`String`	-	-	"Porsche-Factory-Unsecured"
`tagsLastModifiedTimestamp`	`Integer`	-	-	2024-01-08T16:23:19…
`errorFlags`	`Integer`	-	-	0
`rbl` *Remaining battery life*	`Float`	0 - 100	%	26.171875
`temperature`	`Float`	-	°C	17.0
`pressure`	`Float`	-	mbar	1000.0
`vehicleId`	`Integer`	0 - 63	-	7
`wheelPosition` `0 = FL, 1 = FR, 2 = RL, 3 = RR`	`Integer`	0 - 3	-	2
`tyreType`	`Integer`	0 - 63	-	24
`tyreSet`	`Integer`	0 - 63	-	3
`irCoreTemperature1` * *Camera 1 core temperature*	`Float`	-	°C	17.0
`irCoreTemperature2` * *Camera 2 core temperature*	`Float`	-	°C	17.0
`version`	`String`	-	-	1.3.22112301

Field	Format	Range	Units	Example
Sensor firmware version				
`faultCode` *	`Integer`	-	-	0
`pixelCount` *	`Integer`	8 / 28	-	8
`pixelTemperatures` * *Individual IR pixel temperatures*	`Pixels`	-	°C	*see below
`firmwareVersion`	`String`	-	-	1.3.22112301
`moving`	`String`	Yes / No	-	Yes

> **Note**
> Fields marked with an asterisk (*) are only present on IR sensors.

Example


```
{
  "sensorAddress": 31013720310906,
  "sensorName": "1SIR447A",
  "sensorType": "IR",
  "timestamp": "2024-01-18T16:03:25.479937+00:00",
  "receivers": [
    { "name": "receiver1", "signalStrengthInDb": -66 },
    { "name": "receiver2", "signalStrengthInDb": -78 }
  ],
  "sequence": 54,
  "securityCode": "Generic-Factory-Unsecured",
  "tagsLastModifiedTimestamp": "2024-01-08T16:23:19.544+00:00",
  "errorFlags": 0,
  "rbl": 85.546875,
  "temperature": 15.0,
  "pressure": 1640.0,
  "vehicleId": 0,
  "wheelPosition": 1,
  "tyreType": 0,
  "tyreSet": 0,
  "irCoreTemperature1": 16.0,
  "irCoreTemperature2": 16.0,
  "pixelTemperatures": {
    "10": 22.3,
    "13": 22.4,
    "15": 20.2,
    "17": 20.1,
    "19": 21.5,
    "20": 21.9,
    "21": 21.6,
    "22": 22.2
```

```
  },
  "version": "1.3.22112301",
  "faultCode": 0,
  "pixelCount": 8,
  "firmwareVersion": "1.3.22112301",
  "moving": "No"
}
```

Pixel Temperatures

Schema

```
{
  "pixelTemperatures": {
    "pixelNumber": {
      "description": "pixelTemperature",
      "type": "float"
    }
  }
}
```

Example

```
{
  "pixelTemperatures": {
    "10": 22.3,
    "13": 22.4,
    "15": 20.2,
    "17": 20.1,
    "19": 21.5,
    "20": 21.9,
    "21": 21.6,
    "22": 22.2
  }
}
```

Receivers

Schema

```
{
  "receivers": [{ "name": "string", "signalStrengthInDb": "number" }]
}
```

Example

```
[
  { "name": "receiver1", "signalStrengthInDb": -66 },
  { "name": "receiver2", "signalStrengthInDb": -78 }
]
```

Sensor Address

Schema

```
{
  "sensorAddress": {
    "type": "integer",
    "format": "int64"
  }
}
```

Example

```
{
  "sensorAddress": 31013720310906
}
```

> **Note**
>
> The `sensorAddress` is the decimal representation of the sensor's MAC address.

> **Note**
>
> The sensor's serial number can also be calculated from the `sensorAddress` by performing a bitwise `AND` operation with `0xffff`. The serial number is printed on the sensor housing.

Sensor Address Array

Schema

```
[
  {
    "sensorAddress": {
      "type": "integer",
      "format": "int64"
    }
  }
]
```

Example

```
[
  {
    "sensorAddress": 31013720310906
  },
  {
    "sensorAddress": 5278867988801
  },
  {
    "sensorAddress": 5278867988801
  },
  {
    "sensorAddress": 242504952566567
  }
]
```

Tag Info

Schema


```
{
  "sensorAddress": {
    "type": "integer",
    "format": "int64"
  },
  "tagKey": "string",
  "tagType": "string",
  "tagLength": {
    "type": "integer",
    "format": "int64"
  }
}
```


Example


```
{
  "sensorAddress": 5278867988801,
  "tagKey": "Example Tag",
  "tagType": "UTF8",
  "tagLength": 34
}
```

Tag Info Array

Schema

```
[
  {
    "sensorAddress": {
      "type": "integer",
      "format": "int64"
    },
    "tagKey": "string",
    "tagType": "string",
    "tagLength": {
      "type": "integer",
      "format": "int64"
    }
  }
]
```

Example

```
[
  {
    "sensorAddress": 5278867988801,
    "tagKey": "Example Tag 1",
    "tagType": "UTF8",
    "tagLength": 34
  },
  {
    "sensorAddress": 5278867988801,
    "tagKey": "Example Tag 2",
    "tagType": "UTF8",
    "tagLength": 49
  },
  {
    "sensorAddress": 5278867988801,
    "tagKey": "Example Tag 3",
    "tagType": "UTF8",
    "tagLength": 44
  }
]
```

Glossary

API

An application programming interface (API) providing methods to interact with data from/associated with sensors.

DCS

A device communication service (DCS) that handles incoming over-the-air signals from sensors.

SSE

Server-Sent Events (SSE), or data, are sent from the server to the client via an open connection.

self-signed

A self-signed HTTPS certificate is a digital certificate generated and signed by the entity itself, lacking third-party verification

trusted provider

A trusted certificate provider is a reputable third-party entity that issues digital certificates, validating the authenticity of websites and enabling secure HTTPS connections by confirming the identity of the certificate holder.

license request

A file used to obtain a license for the computer running the API.

Swagger page

A webpage providing a visual and interactive representation of the API's design, including routes with example request and response schema.

tag

A plain text value or file that stores customisable information associated with sensor addresses. These are stored virtually by the API in a local database and **not** on the sensors themselves.

Example

```
{
  "tyreType": "wet",
  "tyreSet": 4,
  "notes": ["Uneven wear on inner side", "Sensor low battery"]
}
```

TCP/IP

A transmission control protocol/Internet protocol defines how data is sent and received over a network.

IP address

An Internet Protocol (IP) address is a unique numerical label assigned to each device connected to a computer network.