IIWARI TRACKING SOLUTIONS

IIWARI API SPECIFICATION

25.4.2024

Copyright © Iiwari Tracking Solutions Oy

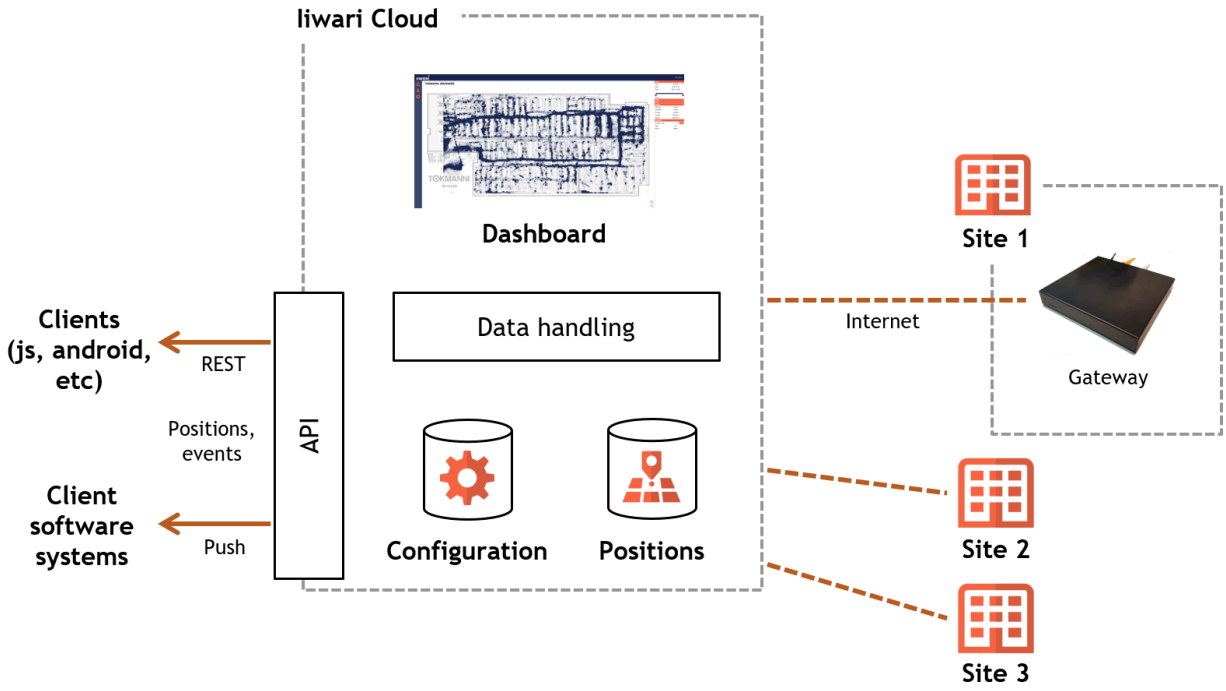
Teemu Lätti teemu.latti@iiwari.com

Table of contents

1. Introduction

This document describes how to use the Iiwari API. The Iiwari API is exposed from the Iiwari cloud (or standalone system) and allows you to use Iiwari positioning system data and features in your own software and applications.

The following figure shows the high-level overview of the Iiwari positioning software system. Each site has a local device “Iiwari Gateway” which manages the local positioning infrastructure, calculates positions and connects to the Iiwari cloud. The Iiwari cloud manages all sites, stores positioning data and exposes interfaces to external systems. External systems can either use Iiwari API (REST interface) or the cloud can push data to them (when such arrangement is agreed). User interface (UI) “Iiwari Dashboard” is available for data visualization. Dash UI is also used to configure sites, floors, zones, assets, tags, users etc.



The user of Iiwari API is a software program (backend, web app, mobile app, etc) which here is referred to as “client”. The Iiwari system serving this API is here referred to as “server”.

All use of Iiwari API should be first agreed with Iiwari Tracking Solutions. In case of any questions, please contact the author.

2. Iiwari API

You can programmatically access the Iiwari API from the following URL:

https://dash.iiwari.cloud/api/v1

In the rest of this document, the base URL is omitted when specifying the URL for operations.

Operations are issued using REST requests. Operations may support multiple HTTP methods (GET, POST, PATCH, DELETE) for different purposes as specified. The request URL may include parameters such as target entity, path parameters and query parameters. For example:

https://dash.iiwari.cloud/api/v1/sites/00000000-0000-0000-0000-000000000000/operation?name=foo&title=bar

Most requests operate with JSON data. Depending on your client you may need to include a header to specify the content type:

```
Content-Type: application/json
```

In this document, JSON objects are specified for relevant content and fields only. You should parse valid JSON from the response body and access only specified fields and ignore others (they may be changed or omitted in the future).

Most database entities are identified by UUID which has the format `"00000000-0000-0000-0000-000000000000"`. Devices (like positioning tags) are identified by HWID (hardware ID) which has the format `"0000-0000-0000-0000"`.

Database entities identified with UUID always have a field “`rev`” which is an automatically increasing number whenever the database entity is changed. Whenever you make a PATCH request to update entity fields you must provide the most recent rev value, otherwise the update request fails. This mechanism is in place to avoid unintended overwriting with old data.

All timestamps are presented in UTC with millisecond precision in the format “`2023-01-01T00:00:00.000Z`”.

Any request which is made using somehow incomplete or invalid payload data, results in a “400 Bad request” response.

Some methods use the WebSocket protocol. Note that in this case request URL has different protocol:

wss://dash.iiwari.cloud/api/v1

2.1. Authentication

All requests must be authenticated (except login). You must specify a valid authentication token in each request you make. The token can be passed as a header:

```
Authorization: Bearer abcdef
```

Alternatively if header cannot be used, you can pass the token as a query parameter:

```
operation?token=abcdef
```

To get an authentication token from your login credentials, you can use the login request. Note that you do not need to make this request every time your application starts, but you can store and reuse the same token. However, if the token becomes invalid for some reason, your application must be able to request the token again. Alternatively you can get the token from Dash UI, see chapter “CURL examples”.

Login request is made with the same credentials (email + password) as for the Dash UI. During development you can use your own credentials, but for production application development we recommend you request from us new specific user credentials for that purpose with only required permissions.

Method:	POST
URL:	`/users/login`
Body:	`{ “email”: "foo@bar.com", “password”: "foobar" }`
Response codes:	200 OK 400 Bad Request 401 Unauthorized

Response:	<pre>{
	  "user": {
	  "id": "00000000-0000-0000-0000-0000000000000",
	  "email":"foo@bar.com",
	  "name":"Foo bar",
	  },
	  "token":"abcdef"
	}</pre>

You can test whether a previously received token is still valid by making a login request with the attached token and no payload. Response 200 OK means the token is still valid.

Allowed sites and operations for a specific user are based on their permissions. Attempting to execute an unallowed operation results in a “403 Forbidden” response.

2.2. Get sites

You can get a list of sites using this call:

Method:	`GET`
URL:	`/sites`
Response:	`[ { "id":"00000000-0000-0000-0000-000000000000", "name":"Foobar" } ]`

The response is a JSON array of sites which you are allowed to see. You can use the site ID to refer to a site in other operations which require it. Alternatively you can get the site ID from Dash UI.

2.3. Get site

You can get detailed data for a specific site using this call:

Method:	`GET`
URL:	`/sites/SITEID`
Response:	`{ "id":"00000000-0000-0000-0000-000000000000", "name":"Iiwari Kuopio", "floors":[{ "id":"00000000-0000-0000-0000-000000000000", "name":"3rd floor", "z_min":500, "z_max":1000, "floormaps":[{`

```
    "id":"00000000-0000-0000-0000-000000000000",
    "position0":{"x":0,"y":0},
    "position1":{"x":100,"y":100},
    "pixel0x":0,
    "pixel0y":0,
    "pixel1x":4000,
    "pixel1y":2000,
    "image_hash":"00000000"
  }],
  "zones":[{
    "id":"00000000-0000-0000-0000-000000000000",
    "name":"Zone 1",
    "type":104,
    "corners":[
      {"x":100,"y":0},
      {"x":200,"y":0},
      {"x":200,"y":100},
      {"x":200,"y":100}
    ]},
  }],
  "pois":[{
    "id":"00000000-0000-0000-0000-000000000000",
    "x":600,
    "y":400,
    "z":100,
    "text":"Poi 1",
    "type":0,
    "state":0,
    "images":[]
  }],
  "assets":[{
    "id":"00000000-0000-0000-0000-000000000000",
    “rev”:1,
    "type":17,
    "tag_hwid":"0000-0000-0000-0000",
    "name":"Asset 1",
    "icon":null,
    "data":{ "foo":"bar" }
```

	<pre> }] }] }</pre>

2.4. Stream

You can continuously receive positions and events from a site using this call:

Method:	`WSS: GET -> WEBSOCKET`

URL:	`/sites/SITEID/stream?filter=kalman`
	
	`/sites/SITEID/stream?filter=kalman&startAt=2023-01-01T00:00:00.000Z&endAt=2023-01-01T00:10:00.000Z`
	
	`/sites/SITEID/stream?filter=kalman&format=geo`
	
	`/sites/SITEID/stream?events=20,21`

Time range (UTC) can be specified but is optional. If “startAt” is specified, the server fetches positions from the database starting at that time. If “endAt” is not specified, the server continues sending real time positions as they arrive. When the stream has finished fetching positions from the database and switches to real time or only specifies realtime, a special “mark” message is sent to indicate this. Applications can use this information to know when history has loaded.

Kalman filter selection is optional and if omitted raw positions are returned. It is recommended that you enable the Kalman filter, because the positioning system is calibrated to work best with it. Omit Kalman filter only in the case you want to do your own filtering.

The parameter “events” is a list of event types. If specified, only those events are returned.

The request is always upgraded to a websocket connection. You can keep the websocket open as long as needed. You need to be able to reconnect if the connection breaks up for some reason.

Possible messages which you will receive from the websockets are specified in the table below. You should verify that the received message is in the expected format and ignore other messages. Field “type” identifies the event type and should be checked first. There will be more events with different payloads in the future.

Field “ts” is the timestamp. The time is in UTC.

Field “node” is the tag HWID which can be used to identify the tag (or other device) when relevant.

Position fields (x, y, z) are in cm relative to the site origin position. Z is cm relative to the floor level of the first floor. If you specify “format=geo” in the call, positions are converted to geo positions (latitude,longitude) assuming the site has been properly configured.

When the position is inside a privacy zone, no coordinates are returned (only timestamp). You can use this as an indication that the tag’s exact position should not be shown. This message can still be used to indicate that the tag is alive and somewhere within the site.

Zone, floor and site events are triggered when a tag enters/leaves a zone/floor/site which is configured in a site in the Dash UI.

Events which are notifications for real time events (such as site change) are not returned when receiving from history.

Real time stream starts	<pre>{
	  "mark":1
	}</pre>
	Streaming events from history has completed and messages after this are real time.

Tag position (local coordinates)	<pre>{
	  "type":0,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000",
	  "x":100,
	  "y":100,
	  "z":100
	}</pre>
	Tag position in local coordinates as cm (this is the default). Tag is identified by HWID as the field “node”.

Tag position (geo)	<pre>{
	  "type":0,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000",</pre>

	<pre>"latitude":64.140161,
	"longitude":28.268121
	}</pre>
	Tag position in geo coordinates (only if requested).

Tag position in privacy zone	<pre>{
	"type":0,
	"ts":"2023-01-01T00:00:00.000Z",
	"node":"0000-0000-0000-0000"
	}</pre>
	Tag moves in a privacy zone. No exact position.

Tag button press	<pre>{
	"type":10,
	"ts":"2023-01-01T00:00:00.000Z",
	"node":"0000-0000-0000-0000"
	}</pre>
	User presses the tag button.

Tag switch on	<pre>{
	"type":14,
	"ts":"2023-01-01T00:00:00.000Z",
	"node":"0000-0000-0000-0000"
	}</pre>
	User switches the tag on.

Tag switch off	<pre>{
	"type":13,
	"ts":"2023-01-01T00:00:00.000Z",
	"node":"0000-0000-0000-0000"
	}</pre>
	User switches the tag off.

Device status	<pre>{
	"type":9,
	"ts":"2023-01-01T00:00:00.000Z",</pre>

	<pre>"node":"0000-0000-0000-0000",
	"value":17
	}</pre>
	Devices on site periodically send this “I’m alive” message. Value is product type which have one of the following values:
	1 = GW
	16 - 18 = MBS
	32 - 35 = BS
	48 - 53 = TAG

Device battery voltage	<pre>{
	"type":1,
	"ts":"2023-01-01T00:00:00.000Z",
	"node":"0000-0000-0000-0000",
	"value":4632
	}</pre>
	Device voltage is its internal battery voltage. Divide the field “value” with 1000 to get battery voltage in V.

Device temperature	<pre>{
	"type":2,
	"ts":"2023-01-01T00:00:00.000Z",
	"node":"0000-0000-0000-0000",
	"value":2245
	}</pre>
	Device temperature is the ambient temperature around the device. Divide the field “value” with 100 to get temperature in Celsius degrees.

Zone enter	<pre>{
	"type":20,
	"ts":"2023-01-01T00:00:00.000Z",
	"node":"0000-0000-0000-0000",
	"zone":"00000000-0000-0000-0000-000000000000"
	}</pre>
	Tag enters a zone.

Zone leave	<pre>{
	  "type":21,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000",
	  "zone":"00000000-0000-0000-0000-000000000000"
	}</pre>
	Tag leaves a zone.

Floor enter	<pre>{
	  "type":24,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000",
	  "floor":"00000000-0000-0000-0000-000000000000"
	}</pre>
	Tag enters a floor.

Floor leave	<pre>{
	  "type":25,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000",
	  "floor":"00000000-0000-0000-0000-000000000000"
	}</pre>
	Tag leaves a floor.

Site enter	<pre>{
	  "type":22,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000"
	}</pre>
	Tag enters the site.

Site leave	<pre>{
	  "type":23,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000"
	}</pre>

	Tag leaves the site. Happens when the tag has not been seen for 2 minutes 30 seconds.

Zones status	<pre>{
	  "type":29,
	  "ts":"2023-01-01T00:00:00.000Z",
	  "node":"0000-0000-0000-0000",
	  "zone":"00000000-0000-0000-0000-000000000000",
	  “in_time”:"2023-01-01T00:00:00.000Z",
	  “in_duration”:15000
	}</pre>
	Optional message which is sent periodically. This message is sent separately for each tag and zone and indicates that the tag is currently in the zone. The interval is specified as seconds in the query parameter, for example “followZones=5”. If not specified or has value zero, this message is not sent. This message is not sent from history.

Site change	<pre>{
	  "type":99,
	  "ts":"2023-01-01T00:00:00.000Z"
	}</pre>
	Site has been changed. Triggered by a change to floors, floormaps, zones, pois or assets.

2.5. History

You can get tag positions and events from the history for a site using this call:

Method:	`GET`
URL:	`/sites/SITEID/history?filter=kalman&startAt=2023-01-01T00:00:00.000Z&endAt=2023-01-01T00:10:00.000Z`

This returns positions and events as a single JSON array in the response body instead of websocket stream. Request parameters and response messages are the same as for the stream.

2.6. Locations stream

If you want to receive only positions from a site, you can use this call. Request parameters and response messages are the same as for the stream.

Method:	`WSS: GET -> WEBSOCKET`
URL:	`/sites/SITEID/locations/stream?filter=kalman`

2.7. Locations

If you want to get only positions from the history for a site, you can use this call. Request parameters and response messages are the same as for the stream.

Method:	`GET`
URL:	`/sites/SITEID/locations?filter=kalman&startAt=2023-01-01T00:00:00.000Z&endAt=2023-01-01T00:10:00.000Z`

2.8. Events stream

If you want to receive only events from a site, you can use this call. Request parameters and response messages are the same as for the stream.

Method:	`WSS: GET -> WEBSOCKET`
URL:	`/sites/SITEID/events/stream`

2.9. Events

If you want to get only events from the history for a site, you can use this call. Request parameters and response messages are the same as for the stream.

Method:	`GET`

URL:	`/sites/SITEID/events?startAt=2023-01-01T00:00:00.000Z&endAt=2023-01-01T00:10:00.000Z`

2.10. Tags in site

You can get a list of tags currently in the site using this call:

Method:	`GET`
URL:	`/sites/SITEID/tags`

The returned data is a JSON array containing information for each tag on site including zones where the tag is currently located.

Response:

```
[ {
  "hwid":"0000-0000-0000-0000",
  “floor_id”:”00000000-0000-0000-0000-000000000000”,
  "position":{
    "ts":"2023-01-01T00:00:00.000Z",
    "x":100,
    "y":100,
    "z":100
  },
  “zones”:[{
    “id”:”00000000-0000-0000-0000-000000000000”,
    "name":"Zone",
    “type”:101,
    “in_time”:"2023-01-01T00:00:00.000Z",
    “in_duration”:15000
  }],
  "status_ts":"2023-01-01T00:00:00.000Z",
  “firmware”:”5.6.2”,
  “voltage”:3000
} ]
```

2.11. Tags in floor

You can get a list of tags currently in a specified floor using this call:

Method:	`GET`
URL:	`/sites/SITEID/floors/FLOORID/tags`

The returned data is a JSON array containing information for each tag in the floor including zones where the tag is currently located.

Response:

```
[ {
  "hwid":"0000-0000-0000-0000",
  "position":{
    "ts":"2023-01-01T00:00:00.000Z",
    "x":100,
    "y":100,
    "z":100
  },
  “zones”:[{
    “id”:”00000000-0000-0000-0000-000000000000”,
    "name":"Zone",
    “type”:101,
    “in_time”:"2023-01-01T00:00:00.000Z",
    “in_duration”:15000
  }],
  "status_ts":"2023-01-01T00:00:00.000Z",
  “firmware”:”5.6.2”,
  “voltage”:3000
} ]
```

2.12. Tags in zone

You can get a list of tags currently in a specified zone using this call:

Method:	`GET`
URL:	`/sites/SITEID/zones/ZONEID/tags`

The returned data is a JSON array containing information for each tag in the zone.

Response:

```
[ {
  "hwid":"0000-0000-0000-0000",
  "position":{
    "ts":"2023-01-01T00:00:00.000Z",
    "x":100,
    "y":100,
    "z":100
  },
  "status_ts":"2023-01-01T00:00:00.000Z",
  “firmware”:”5.6.2”,
  “voltage”:3000
} ]
```

2.13. Zones with tags

You can get a list of site zones including tags currently in the zones using this call:

Method:	`GET`
URL:	`/sites/SITEID/zones/tags`

The returned data is a JSON array containing information for each zone on site and a list of tags currently in the zone. Field “in_time” tells when the tag has entered the zone. Field “in_duration” tells the number of milliseconds the tag has been in the zone.

Response:

```
[ {
  “id”:”00000000-0000-0000-0000-000000000000”,
  "name":"Zone",
  “type”:101,
  “tags”: [ {
    "hwid":"0000-0000-0000-0000",
    “in_time”:"2023-01-01T00:00:00.000Z",
```

	<pre>"in_duration":15000
	}]
	}]</pre>

2.14. Tag status

You can get the current status of an individual tag using this call:

Method:	GET
URL:	/tags/hwid/0000-0000-0000-0000/status

The returned data is a JSON object containing tag information. The current position is identified by fields “site_id” and “floor_id” and coordinates. List of zones is returned in which the tag is currently. For each zone, the time when the tag has entered the zone is given in the field “in_time”. Field “in_duration” tells the number of milliseconds the tag has been in the zone.

Response:	<pre>{
	"hwid":"0000-0000-0000-0000",
	“site_id”:”00000000-0000-0000-0000-000000000000”,
	“floor_id”:”00000000-0000-0000-0000-000000000000”,
	"position":{
	"ts":"2023-01-01T00:00:00.000Z",
	"x":100,
	"y":100,
	"z":100
	},
	“zones”:[{
	“id”:”00000000-0000-0000-0000-000000000000”,
	"name":"Zone",
	“type”:101,
	“in_time”:"2023-01-01T00:00:00.000Z",
	“in_duration”:15000
	}],
	"status_ts":"2023-01-01T00:00:00.000Z",
	“firmware”:”5.6.2”,</pre>

	<pre> “voltage”:3000 }</pre>

2.15. Search

You can search assets (tags) from a site using this call:

Method:	`GET`

URL:	`/sites/SITEID/assets/search`
	
	`/sites/SITEID/assets/search?asset_types=1,3`
	
	`/sites/SITEID/assets/search?hwid=0000-0000-0000-0000`
	
	`/sites/SITEID/assets/search?name=Foo`
	
	`/sites/SITEID/assets/search?data=field:value`

This makes a search to all site assets and any global assets which are currently in the site. You can specify any number of parameters specified above. The search is a filtering search where all specified parameters must match for any given asset. If no parameters are given, the result is all assets.

The returned data is a JSON array of matching assets:

Response:	<pre>[{
	  "id":"00000000-0000-0000-0000-000000000000",
	  “rev”:1,
	  "site_id":"00000000-0000-0000-0000-000000000000",
	  "type":1,
	  "tag_hwid":"0000-0000-0000-0000",
	  "name":"Foo",
	  "icon":”...”,
	  "data":{ "field":"value" },
	  "tag":{
	  "id":"00000000-0000-0000-0000-000000000000",</pre>

	"hwid":"0000-0000-0000-0000",
	"product":50,
	"fw_version":"1.0.0",
	"status_at":"2023-01-01T00:00:00.000Z",
	"voltage":4632,
	"z":100
	},
	"position":{
	"ts":"2023-01-01T00:00:00.000Z",
	"x":100,
	"y":100,
	"z":100
	}
	}]

The position field may be null if the tag is not currently detected on the site. Make sure to validate the position timestamp since the position can be old (last known position).

2.16. Change asset

Tag attributes can be changed by creating a site specific asset for it. You can create a new asset using this call. You must specify a tag HWID and there can be only one asset per HWID in a site.

Method:	POST
URL:	/assets

Data for POST:	{
	"site_id":"00000000-0000-0000-0000-0000000000000",
	"tag_hwid":"0000-0000-0000-0000",
	"type":17,
	"name":"Asset 1",
	"data":{ "foo":"bar" }
	}

If you want to change existing asset attributes, use the following call and specify the asset ID in the url. Revision must match with the latest entity version. When updating, tag HWID cannot be

changed. When updating, only the specified fields in the custom data field will be overwritten and others will be left intact.

Method:	`PATCH`
URL:	`/assets/ASSETID`
Data:	`{ "rev":1, "type":17, "name":"Asset 1", "data":{ "foo":"bar" } }`

You can change the asset icon using this call. The content is multipart form data where a field named “image” contains a binary PNG/JPEG image. The image size must not exceed 512x512 pixels.

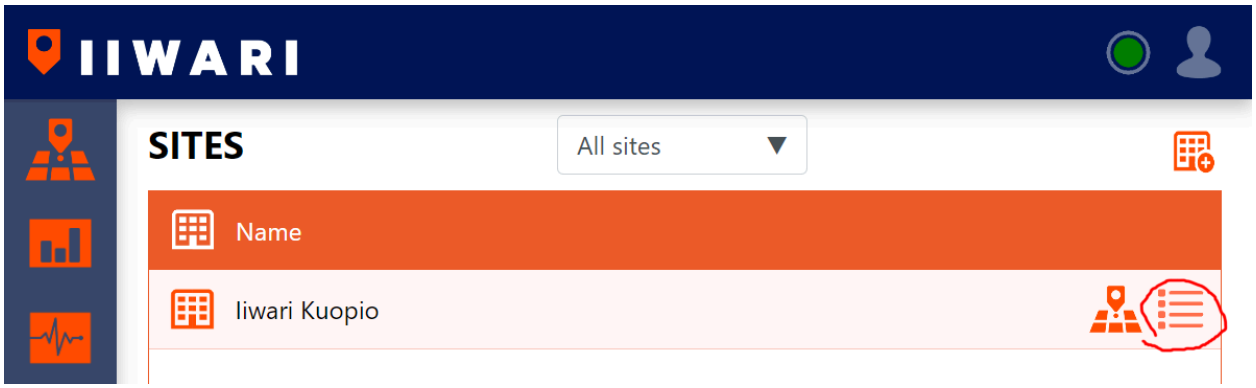
Method:	`PATCH`
URL:	`/assets/ASSETID/icon`
Data:	`Form data, field “image”`

Existing asset icon can be removed with this call.

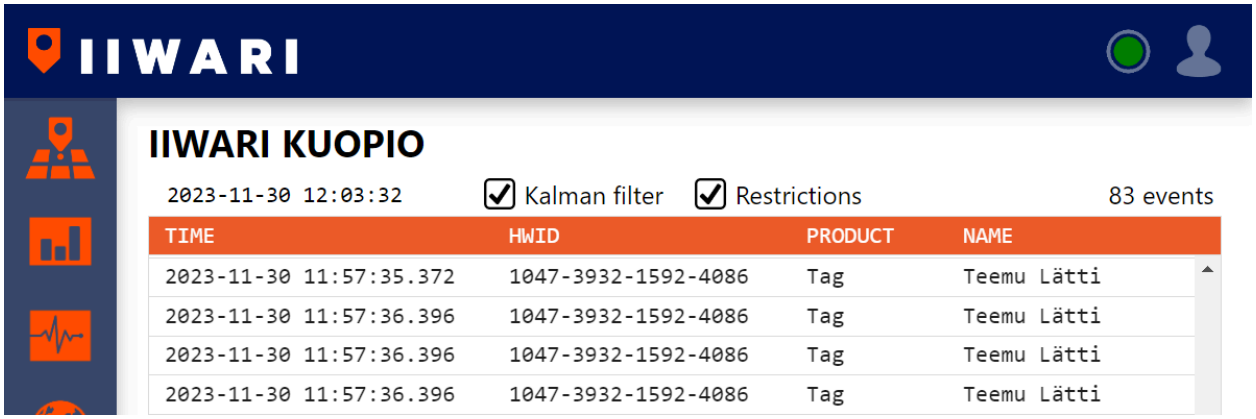
Method:	`DELETE`
URL:	`/assets/ASSETID/icon`

3. Dash UI

To explore and debug events, you can use the Dash UI which shows the same messages which you would programmatically receive from the websocket. In Dash, click the right side icon to open the events page (instead of clicking the site name which opens the map).



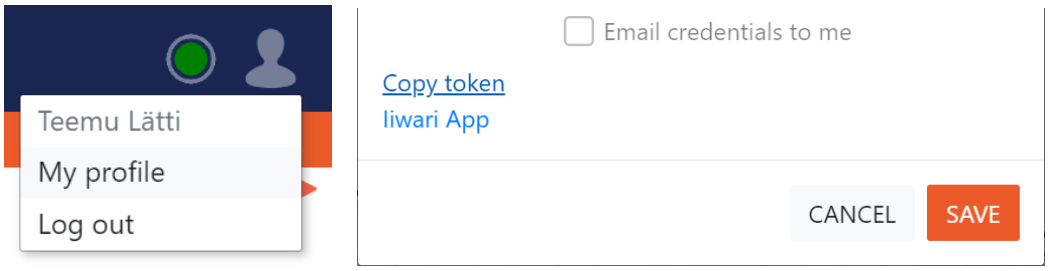
The events page can be used to view events in real time or you can select a time range from history by clicking the time range on top of the page.



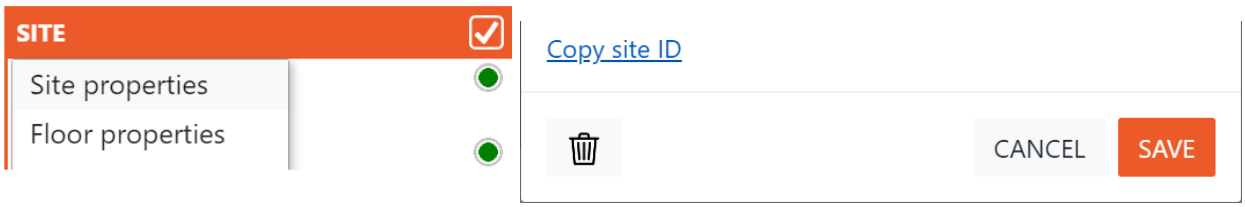
4. CURL examples

You can test the Iiwari API if you have code or a tool which is capable of making HTTP REST requests. This chapter describes how to use the Linux command prompt tool CURL in your own computer to access the Iiwari API.

To access the API, you first need a valid authorization token. You can use your personal token which is connected with your login credentials. Login to Dash at https://dash.iiwari.cloud/ and click the user icon top-right and select “My Profile” from the menu. Then in the dialog, select “Copy token”.



Most methods require a site ID which refers to a site which the methods are run against. Open Dash, open your site, right-click on the “SITE” drawer on the right or the site title in top-left and select “Site properties” from the menu. Then in the dialog, select “Copy site ID”.



For example, to start receiving locations and events, you can use the following command to open a websocket connection. Replace TOKEN with the token copied above, and replace SITEID with the site ID of a site that you have access permission for.

```
curl -H "Authorization: Bearer TOKEN" -H "Connection: Upgrade" -H
"Upgrade: websocket" -H "Sec-WebSocket-Key: MDEyMzQ1Njc4OUFCQ0RFRg==="
-H "Sec-WebSocket-Version: 13" --no-buffer --http1.1 -s -o -
"https://dash.iiwari.cloud/api/v1/sites/SITEID/stream?filter=kalman"
```

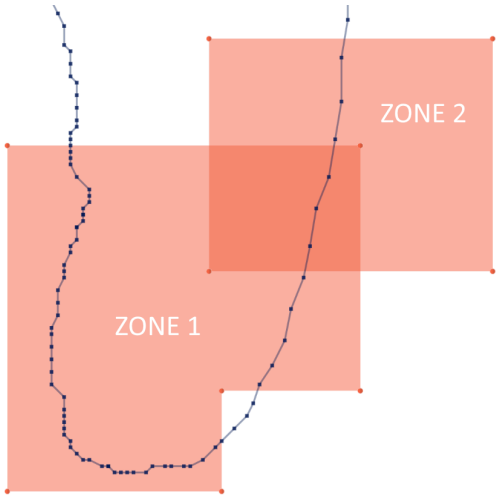
The following example shows how to access a method which is a simple GET call with parameters as part of the URL.

```
curl -H "Authorization: Bearer TOKEN" -s -o -
"https://dash.iiwari.cloud/api/v1/sites/SITEID/history?filter=kalman&
startAt=2023-01-01T12:00:00Z&endAt=2023-01-01T12:01:00Z"
```

Using the format of these examples, you can make a request for any of the methods specified in this document.

5. Zones

Zones are virtual areas which can be configured in the Iiwari positioning system to trigger zone events, gather analytics data or restrict positioning. Zones are configured in the Dash UI. Zones do not need to be rectangular in shape but can have any number of corners. Zones can overlap as they work independently from each other.



5.1. Zone types

Each zone has a type which affects how it is applied and used. Most applications use the normal zone type and use application internal logic to distinguish between zones (using zone IDs).

Normal zone (zone type 0)	Normal zone type used for zone events and analytics.

There are special zone types which can be used to restrict the behavior of the positioning system in certain areas in some ways. These are called restriction zones. Restriction zones can be configured using the Dash UI.

Since restriction zones have an effect on tag position, the resulting positions are used in Dash UI, Dash API (stream) and zone service. Restriction zones are applied after the Kalman filter but before other logic. Note that restriction zones do not generate zone events and are not returned in zone status API calls.

Include (zone type 2)	If there are any include zones configured for a floor, the tag position must be inside one of them. Otherwise, the position is ignored as it never happened. These types of zones can be used to restrict the positioning only to cover certain areas and completely ignore positions outside them, for example the interior of a building.
Force include (zone type 3)	Force include zones behave similarly as include zones, but instead of ignoring a position when outside, it is moved to the nearest position inside the zone. These types of zones can be used to force positions away from areas of challenging radio environments.
Exclude (zone type 1)	When a tag position is inside any configured exclude zone, its position is ignored as it never happened. These types of zones can be used to exclude certain areas from positioning, for example because of a challenging radio environment or privacy reasons.
Privacy (zone type 4)	Privacy zone(s) can be used to hide the exact position of a tag. When a tag is inside any of the privacy zones, its position is reported to be in site but without coordinates. This type of zone can be used to exclude certain areas from positioning for privacy reasons.

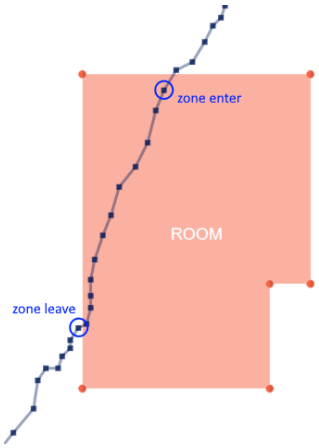
5.2. Zone events

Zone events are automatically generated by the zone service component which is an integral part of the Iiwari positioning system. Your application can use the event stream to follow these events and determine in which zone(s) a specific tag is currently located without the need to calculate from coordinates since the Iiwari positioning system maintains a live status of all tags in all zones for you.

In addition to zone events, you can use Dash API methods to fetch the current status of tags and zones. Either to fetch by zone(s) which includes a list of tags currently in them, or to fetch by tag(s) which includes a list of zones where they are currently located. See chapters 2.10 - 2.14.

Zone events are stored in the history (as are all events). It is possible to fetch them from the history for any time period using Dash API methods. Zones can also be used as parameters in many analytics methods to fetch data.

Whenever a tag enters a zone, an event “zone enter” (type 20) is generated. Whenever a tag leaves a previously entered zone, an event “zone leave” (type 21) is generated. The structure of zone events is specified in chapter 2.4. Both of the zone events specify the time of the event, the tag HWID and the zone ID involved. Events are generated independently for each tag and zone combination regardless if zones overlap.



Dash map

2024-01-18 12:18:48.058	0447-3034-49B0-8828	TAG	Position	3.14	1.95	10.00	
2024-01-18 12:18:48.058	0447-3034-49B0-8828	TAG	Zone enter				Room
2024-01-18 12:18:48.265	0447-3034-49B0-8828	TAG	Position	3.12	2.00	10.00	
2024-01-18 12:18:48.472	0447-3034-49B0-8828	TAG	Position	3.10	2.08	10.00	
2024-01-18 12:18:48.679	0447-3034-49B0-8828	TAG	Position	3.07	2.14	10.00	
2024-01-18 12:18:48.886	0447-3034-49B0-8828	TAG	Position	3.03	2.19	10.00	
2024-01-18 12:18:49.093	0447-3034-49B0-8828	TAG	Position	3.01	2.26	10.00	
2024-01-18 12:18:49.300	0447-3034-49B0-8828	TAG	Position	2.99	2.31	10.00	
2024-01-18 12:18:49.507	0447-3034-49B0-8828	TAG	Position	2.97	2.37	10.00	
2024-01-18 12:18:49.714	0447-3034-49B0-8828	TAG	Position	2.96	2.42	10.00	
2024-01-18 12:18:49.818	0447-3034-49B0-8828	TAG	Position	2.96	2.46	10.00	
2024-01-18 12:18:49.921	0447-3034-49B0-8828	TAG	Position	2.96	2.49	10.00	
2024-01-18 12:18:50.025	0447-3034-49B0-8828	TAG	Position	2.95	2.53	10.00	
2024-01-18 12:18:50.128	0447-3034-49B0-8828	TAG	Position	2.93	2.54	10.00	
2024-01-18 12:18:50.128	0447-3034-49B0-8828	TAG	Zone leave				Room
2024-01-18 12:18:50.232	0447-3034-49B0-8828	TAG	Position	2.91	2.57	10.00	

Dash events page

Zone enter event is triggered at the timestamp of the first position which lies inside the zone. Zone leave event is triggered at the timestamp of the first position which lies outside the previously entered zone.

If a tag timeouts, meaning there has been no position detected for it for the timeout period (by default 2min 30sec), the tag is considered to have left the site. This also triggers zone leave events for all zones it was currently in.

5.3. Zone events filtering

The zone service applies several filters to aid in properly determining tag positions in zones depending on the use case.

Firstly, the tag positions are pre-filtered with the Kalman filter. This removes most erroneous positions which may be created in a challenging radio environment so that they do not affect the zone determination. The Kalman filter parameters are the same as used in the Dash UI by default. It is possible to configure the zone service to use different Kalman filter parameters for a specific use case or by depending on the tag or zone.

Secondly, the zone service applies any configured restriction zones (see chapter 5.1) before using the tag position.

Finally, the zone service filters zone enter/leaves for a specific tag and a zone when they happen rapidly. If only one position lies inside the zone and is immediately followed by a position outside the zone, no events are generated. The same single-position-filtering is done for leaving the zone.

It is also possible to configure the zone service to have a minimum time for a tag to be inside/outside a zone to actually trigger an event, for example 5 seconds. This is useful in use cases where it does not make sense to report very short periods inside/outside a zone.