Inovonics Cloud

Integration Guide

Preface

Trademarks and Copyrights

- EchoStream® is a registered trademark of Inovonics Corporation

Contact Information

- Inovonics Technical Services. support@inovonics.com or 1.800.782.2709.

Revision History

Revision	Name	Description	Date
0.1	Kevin Stoner	Initial Revision	12/22/21
0.2	Kevin Stoner	Accepting Todd's Review	12/30/21
0.3	Kevin Stoner	Added Hardware Part Numbers Table	12/31/21
0.4	Kevin Stoner	Adding Cooper's Review Changes	1/12/22
0.5	Kevin Stoner	Fixing Date	1/12/22
0.6	Kevin Stoner	Updating example URL	1/12/22
0.7	Kevin Hardy	Merged content from various sources	7/19/23
0.8	Kevin Hardy	Including Advanced Location & Widget guides	9/20/2023
0.9	Kevin Hardy	Including MQTT YAML content	10/25/2023
1.0	Kevin Hardy	Including Asset Tags & Staff Badges	4/19/2024
1.1	Kevin Hardy	Resolved navigation & formatting issues	5/10/2024

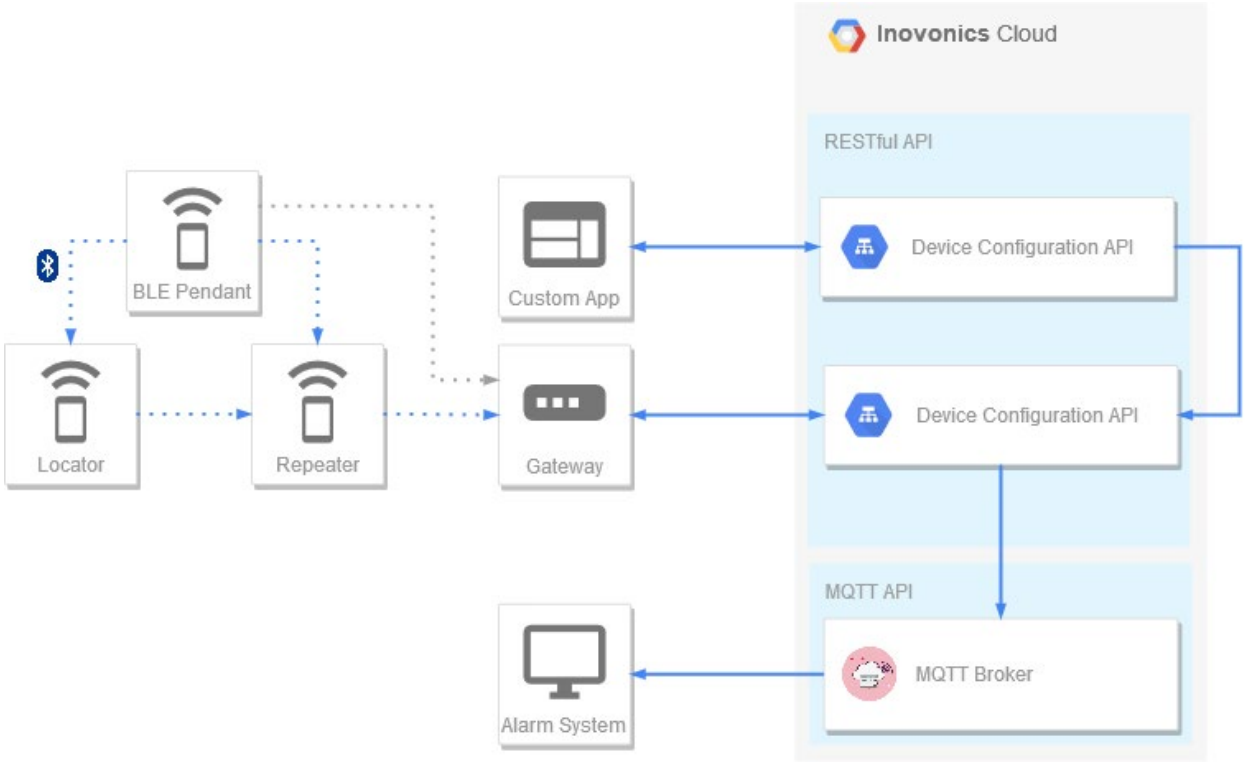
Table of Contents

1 Purpose

This document is intended to provide high level guidance for integrating your application with the Inovonics Cloud Service Ecosystem.

2 System Overview

The Inovonics Cloud system is designed with the objective of allowing the configuration and monitoring of Inovonics devices through the cloud. Some common uses of the system are receiving locations of alarm events, processing of fall detection alarm events and associated algorithm data, and general device health monitoring.



In the above diagram, a custom developed application may leverage RESTful APIs to configure and set up a site with devices in the Inovonics Cloud. Physical devices on each site will communicate wirelessly over the EchoStream® network, which will be heard and sent up to the Inovonics Cloud via a gateway. From there, important events such as alarm locations will be sent to the Inovonics MQTT Broker. A custom MQTT client can then subscribe to the broker and handle incoming events in the appropriate manner. Each of these components will be discussed in further detail later in this document.

3 Hardware Components

3.1 Fixed Transmitters

Fixed transmitters are Inovonics EchoStream® transmitting devices that are designed to remain in a specific location at a site. Examples of fixed transmitters are door sensors and smoke detectors. These devices transmit events over the EchoStream® network which will be collected by the Gateway on the site.

3.1.1 Roaming Devices

3.1.2 Mobile Duress Transmitters

Mobile Duress Transmitters have several form factors, like wearable pendants & fob style buttons. These devices transmit alarms via the EchoStream® network, and also report their location via BLE to Locator devices.

3.1.3 Asset Tags & Staff Badge

Asset Tags & Staff Badges are devices that track location using BLE beaconing. These devices do not transmit via EchoStream®, and instead work in conjunction with the Locator devices. When a BLE message is heard by a Locator, the Locator then converts the message into EchoStream® and sends the device location through the network.

3.2 Locators

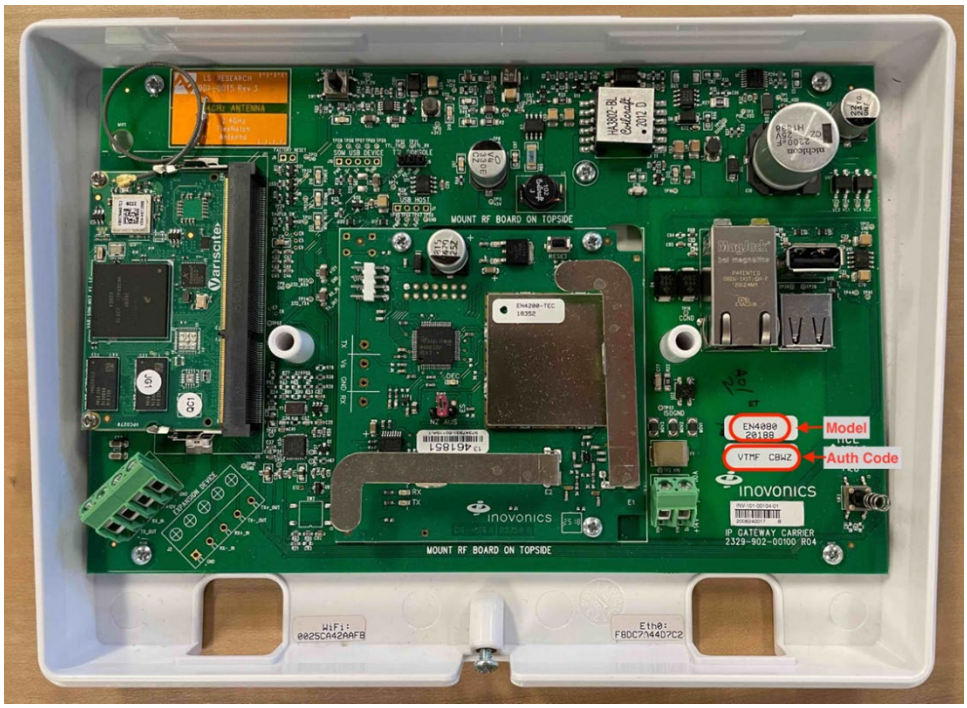
Locators are a BLE to EchoStream® “bridge” that can be installed throughout a site to locate Bluetooth equipped mobile pendants when they alarm. The locator listens for BLE messages from pendants on the site and will convert those messages to EchoStream®. Those messages can be used to determine the closest locator to a mobile pendant when it is activated.

3.3 Repeaters

Repeaters listen for EchoStream® messages and repeat them by re-transmitting the message. They are used for extending the range of the EchoStream® network on the site. To meet certain standards, repeaters on some sites must run in Directed Messaging mode.

3.4 Gateway

The Gateway is the connection between the site and the cloud. The Gateway will listen for, and collect EchoStream® messages using the built in receiver, and will relay them up to the Inovonics Cloud for further processing and storage. In the case of network issues between the gateway and the site, the gateway can store messages until it is able to communicate with the cloud once again. The gateway must be added to a site in the Inovonics Cloud in order to pull site configuration information and send up EchoStream® traffic from devices.

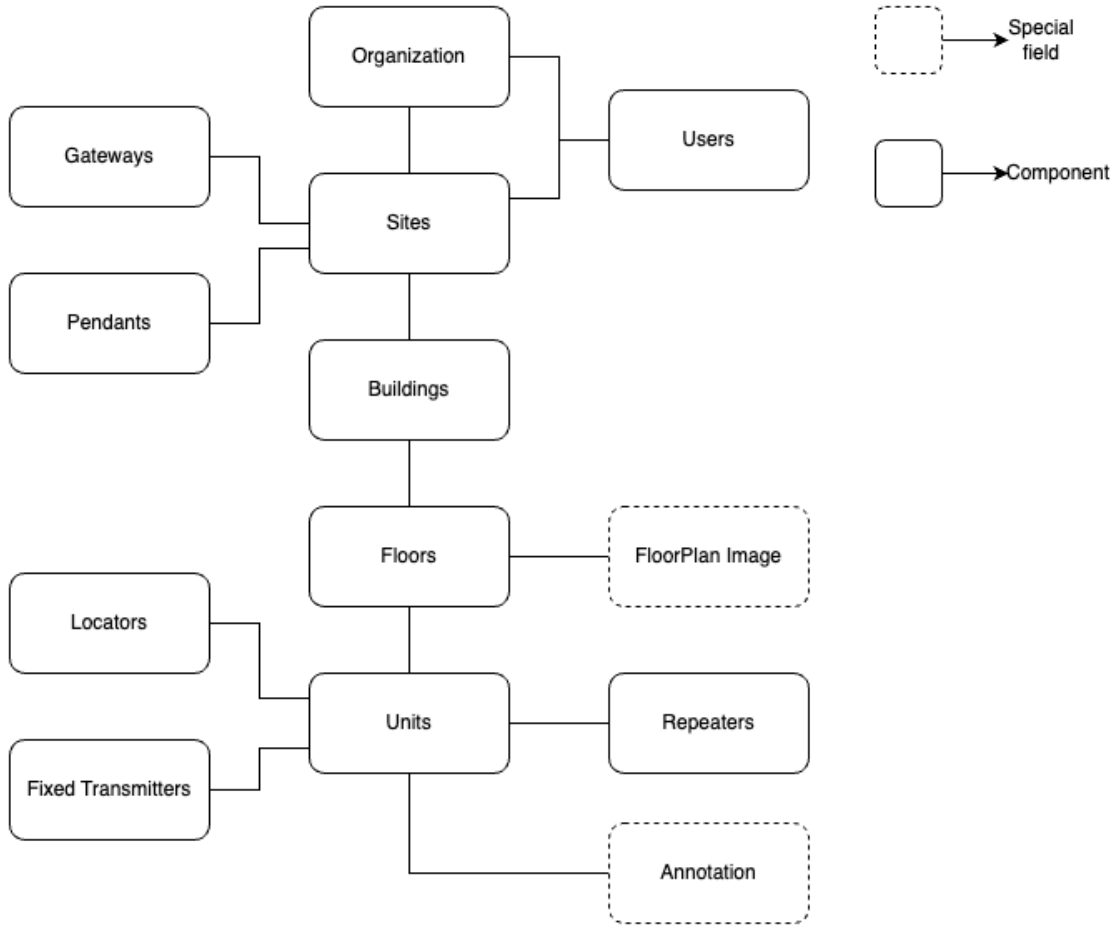


This image shows where the Authorization Code and model can be found on the Gateway device. The serial number and MAC address label can be found on the Gateway device, Gateways built in 2020 and earlier (VP20*, VP19*, VP18*) will have a label with a barcode, and gateways built in 2022 and after (VP22*, VP23*) will have the label without barcode.

Note: Inovonics will add Organization, and customers can add sites, buildings, devices, etc. Inovonics will also create one admin user; additional users are created by customers.

4 Organization Hierarchy

The organization hierarchy as pictured below demonstrates the relationship of different objects within the Inovonics Cloud. When creating new sites, it is recommended to follow this pattern in order to avoid any issues with compatibility.



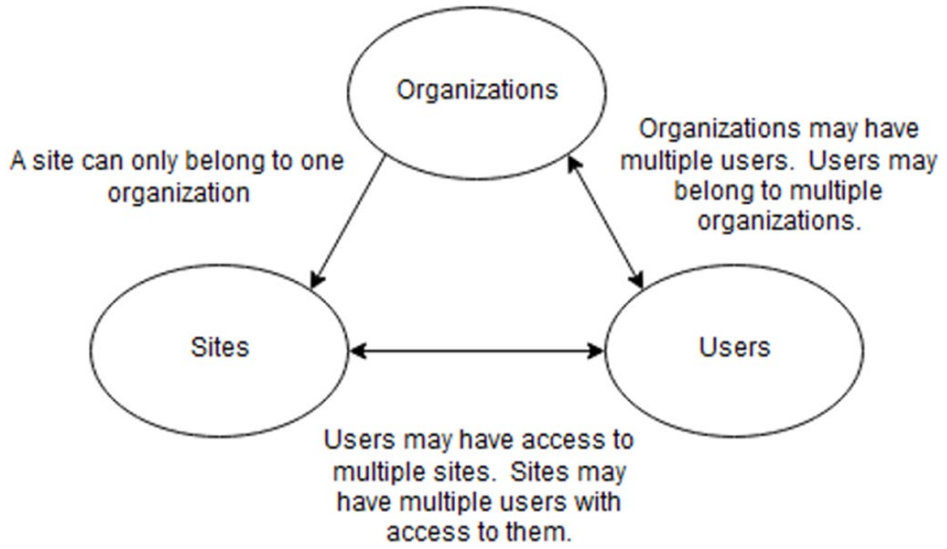
4.1 Organizations

At the topmost level, everything belongs to an organization. An organization can be considered a partner of Inovonics. Organizations are created by Inovonics as well as the first few users within the organization, to get the partner started. From there it is on the partner to create the subsequent users and sites.

4.2 Users

Users are members of the organization that need to create, view, or modify sites. Users are NOT the end customer (ex: senior living community residents, duress pendant carrier). Every user within an organization requires a unique email address, even if the user is a service account user. There are three different permission levels users can be assigned: Administrator, Technician, and Viewer. Administrators are allowed to create and modify sites, as well as administer other users within the organization. Technicians are allowed to create and modify sites but cannot administer users. Viewers

can only view sites but cannot make any changes otherwise. Besides assigning roles, Administrators can optionally restrict access for other users to certain sites over certain timeframes.



4.3 User Types

User Types define permission levels and can be thought of as User Roles.

Administrator: Ability to add and revoke user access at an organizational level. Administrators will be able to assign user access to sites within their organization. Administrators will be able to add and remove users from the organization. Administrators also will have all the capabilities that Technicians and Viewers do.

Technician: Ability to Add/Edit sites within an organization based on the amount of access they are granted by an administrator. Technicians also will have all the capabilities that Viewers do.

Viewer: Ability to view sites based on the amount of access they are granted by an administrator. They cannot modify data in the Inovonics Cloud Services ecosystem.

4.4 Sites

Sites are a specific place where an EchoStream® network of Inovonics devices are installed (e.g. a hospital). Sites may contain many devices, with the only restriction being that each site must only have one gateway.

4.5 Buildings

Buildings can be considered a type of folder to organize the site. A smaller site may only have one building, but larger sites may have multiple buildings. While it is not necessary to leverage buildings, it is recommended to tie locators, repeaters, and fixed transmitters to a building, as it helps to better define the location of each device. Mobile pendants should not be tied to a building as they may move freely across the site.

4.6 Floors

A designated level in a building that identifies the vertical location of a collection of units, e.g., first floor, second floor, third floor, etc.

4.7 Floorplan

A technical drawing (to scale) showing a view from above, of the relationships between rooms, spaces, and other physical features at one level of a structure.

4.8 Units

A Unit is a designated space within a building that represents a specific operating space.

For senior living communities a 'Unit' should be used to designate a resident's apartment (ex: "Unit 208") as well as specific areas like hallways, cafeterias, recreation rooms, and so on.

For other applications a 'Unit' could be used to designate an office suite, warehouse area, or any other specific area where an Inovonics sensor device will be installed.

4.9 Devices

The types of devices on each site are the same as defined in the *Hardware Components* section of this document. When adding devices, each device must first be added directly to the site (POST request). From there to add a device to a Unit, the relationship can be created between a device and Unit within the same site (PUT request). Apart from the gateway, devices must be registered by entering the TXID of the device. This is the number found usually on a sticker on the outside casing of the device. Sites must have a unique set of TXIDs on them, however it is possible to use the same TXID on two separate sites.

5 Integration Options

Inovonics Offers these options for integrations: API, Embedded Widgets, and the Inovonics Cloud Service web front end (ICS Web). All integrations work together and may be used in conjunction with one another.

5.1 API

The API integration allows your application to be the interface with your users while utilizing the features of Inovonics Cloud Services.

- Fully functional
- Total control of your user's experience (UX)
- Standard development cycle to implement features

5.2 Embedded Widgets

The API also enables your application to embed fully functional features into your application, drastically reducing development efforts.

- Embed feature UX + logic in your applications
- Minimal development to implement

5.3 Inovonics Cloud Services Web Application

You may also utilize the fully functional web application Inovonics Cloud Services.

- Turnkey web application
- Zero development to implement
- Inovonics branded & disconnected from partner systems.

6 RESTful API

The RESTful API is used for configuring sites and administering users. Site configuration involves registering the devices that will be used on the site to the cloud. When the configuration is created or changed, it is pushed down to the gateway on the site. Configuration information is also used to send out more detailed information during events and alerts. The RESTful APIs also allows for the administration of users. Users can be given different permission levels and access to be able to view and modify sites within their organization.

6.1 Documentation

API Documentation: Refer to the documentation link under the Documentation section.
Swagger Editor: https://editor.swagger.io/

6.2 REST API Requirements

The URL for making API calls will be provided by Inovonics once the Organization is set up for a partner. All API requests require that a user is authenticated via OAuth 2 and that the connection is secured by TLS. Users making requests must have the correct access and permissions to the resource they are requesting. For instance, a Viewer may not make a request to rename a site. Upon authenticating with the API, the user will be provided with a Bearer token that can be used in subsequent requests to verify their authentication.

6.3 Request Format

With a few exceptions, every API request (and response) will be in JSON format. To make a request, it is required to specify the content-type of the body (usually “application/json”) in the headers of the request. There are a few API requests that accept no request body as well as an import API that accepts spreadsheets. More information about these requests can be found in the API documentation. The authentication (bearer) token should also be placed in the header of each request. Every API endpoint in the Inovonics Cloud ends in a forward slash (“/”) character, if the character is left off an error will be returned. In general, HTTP verbs are used to perform the associated commands. To add a new site, the user would send a POST request. To remove the site, the user would send a DELETE request. To modify the site, the user would send a PUT request. To retrieve information about a site, the user would send a GET request.

Example Request to Add Site:

```
Request:

POST https://security-api.inovonics.com/v1/organizations/<organization_id>/sites/
```

```
Headers:

Authentication: Bearer <access key>
Content-Type: application/json
```

```
Body:

{
    "name": "a new site",
    "code": "ans",
           "timezone": "US_EASTERN"
}
```

6.4 Response Format

The response to a request also follows standard HTTP protocol. For example, if a resource does not exist, a 404 NOT FOUND response will be returned. Other common error codes returned are 409 CONFLICT for attempting to create something that already exists (e.g., a transmitter with a duplicate TXID), 403 FORBIDDEN, 401 UNAUTHORIZED, and 400 INVALID DATA for attempting to make a poorly formatted request. For valid requests, a 200, 201 or 204 may be returned. A 201 indicates that something new was created in the case of a POST request, and a 204 indicates that there is no return body in the response. Please refer to the API documentation for a complete list of request and response bodies and codes.

Example Response to Add Site:

```
Response:
201 Created
```

```
Body:
NOTE: Some fields have been left out of this example for brevity.
{
    "address": "",
    "code": "abc",
    "name": "a new site",
    …
    "site_id": "<site_id>",
    "timezone": "US_EASTERN"
}
```

6.5 Referencing Objects

Upon creation, all objects such as users, sites, devices, and buildings will generate a unique ID which is returned in the response. The ID is formatted as a GUID and should be used to reference each object in subsequent API requests. This ID should not be confused with the TXID of a device which is a separate ID used to tie a physical device to its cloud object representation.

6.6 Enum APIs

There are a few APIs that exist for the sole purpose of providing values that can be entered into other APIs. For example, a developer might call the transmitter model API to find what values they can use for the model of a new transmitter device.

6.7 Import/Export APIs

As an alternative to configuring a site with individual APIs, a user may leverage the import/export APIs to import in an Excel spreadsheet containing an entire site's configuration. To do this, the site must be created first using a POST command. An export spreadsheet may then be created by doing a GET on `/sites/<site_id>/export`. Once changes have been manually applied to the spreadsheet, it can be re-imported with a PUT on `/sites/<site_id>/import/`.

6.8 Report APIs

A few report APIs exist for the purpose of pulling historical data or snapshots in time. These reports should not be leveraged for real time event handling. Instead, the MQTT API should be used to respond to device events.

6.9 Creating Relationships APIs

All objects such as devices, buildings, floors and units must be added to the site initially with a POST request. After adding the object to the site, further relationships can be created by performing PUT requests between two objects. For example when adding locators, each locator must first be added directly to the site (POST request on /sites/<site_id>/transmitters/). From there in order to add a locator to a unit, the relationship can be created between a device and unit within the same site (PUT request on /sites/<site_id>/units/<unit_id>/transmitters/<transmitter_id>/).

6.10 Common API endpoints

All Inovonics Cloud Service integrations will rely on the foundations outlined below. This guide assumes that your Organization has already been created.

Please note that the top-level Organization can only be created by the Inovonics Technical Service team.

6.10.1 Create a site

Make a POST API call to the endpoint: `/organizations/{organization_id}/sites/` with the required headers and body.

6.10.2 Add a gateway to the site

Using the above site_id add a gateway to this site using a POST API call to the endpoint: `/sites/{site_id}/gateways/` with the required headers and body.

6.10.3 Add a building to the site

Using the above site_id add a building to this site using a POST API call to the endpoint: `/sites/{site_id}/buildings/` with the required headers and body.

6.10.4 Add a floor to the site

Using the above site_id add a floor to this site using a POST API call to the endpoint: `/sites/{site_id}/floors/` with the required headers and body.

6.10.5 Add a unit to the site

Using the above site_id add a unit to this site using a POST API call to the endpoint: `/sites/{site_id}/units/` with the required headers and body.

6.10.6 Add a device to the site

Using the above site_id add a device (transmitter) to this site using a POST API call to the endpoint: `/sites/{site_id}/transmitters/` with the required headers and body.

Use the endpoints below to configure the correct hierarchy & relationships between these elements.

6.10.7 Associate a floor to a building

To add a floor to a building, make a PUT API call to the endpoint: `/sites/{site_id}/buildings/{building_id}/floors/{floor_id}/` with the required headers. This will associate the given floor to the given building.

6.10.8 Associate a unit to a floor

To add a unit to a floor, make a PUT API call to the endpoint: `/sites/{site_id}/floors/{floor_id}/units/{unit_id}/` with the required headers. This will associate the given unit to the given floor.

6.10.9 Associate a device to a unit

To add a device to the unit, make a PUT API call to the endpoint: `/sites/{site_id}/units/{unit_id}/transmitters/{transmitter_id}/` with the required headers. This will associate the given device to the given unit.

6.11 MQTT Feed

To receive location and fall information for alarms as well as other device notifications, the Inovonics MQTT Feed can be leveraged. Clients that are subscribed to the MQTT broker will receive events and notifications as the cloud processes them without having to poll the RESTful API for information, as this is a more efficient information exchange mechanism.

Please refer to Appendix B for the YAML content for the MQTT Feed

6.11.1 MQTT Integration Setup

MQTT clients do not use the same authentication as the RESTful API, the authorization credentials (username/password) must be configured by creating an MQTT integration. There are two ways that MQTT integrations can be set up to allow a client to listen to MQTT traffic: at the site level, or at an organization level. MQTT integrations at the site level will allow for a unique set of credentials to be assigned to each site. These credentials can be used to connect and subscribe to MQTT traffic for a specific site. Alternatively, an organization level integration creates a single set of credentials for the entire organization that can listen to traffic for all sites within the organization. Only one type of integration may be used at a time per organization, so either each site must have an individual integration or the organization itself must have one integration, but not both. Each type of integration can be set up with RESTful APIs (see REST API documentation). There are unique API calls for creating an organization level integration and site level integration respectively.

6.11.2 Connecting to the MQTT Broker

The MQTT broker URL will be provided by Inovonics upon requesting to do an MQTT integration. The broker allows for both TLS encrypted and unencrypted connections, however it is highly recommended to use an encrypted connection for production sites. Port details are available from the Inovonics Engineering team upon request.

The Inovonics broker has a CA signed certificate that is used for encrypted connections. Currently the Inovonics MQTT Broker uses MQTT version 5. To connect, the username and password configured in the MQTT integration will need to be used. The client ID for the connection must be globally unique so it is recommended to use a random string of characters for the client ID. In the future there may be further restrictions on the client IDs that can be used. When connecting it is also recommended to set the “Clean Session” flag to False. This will allow a client that disconnects to reconnect and receive any messages that it missed. Currently the Inovonics MQTT broker will attempt to store unsent messages for up to 1 day, but this may vary significantly based on the broker storage that is available.

6.11.3 MQTT Subscriptions

MQTT Subscriptions can be made after connecting with the client. In general, Inovonics recommends subscribing to MQTT topics with a QOS of 2 to prevent missing or duplicate messages. Currently there is a common “status” topic that can be leveraged for determining the health of the cloud. Every authenticated MQTT user automatically has access to this topic. The rest of the topics are restricted by MQTT username. This prevents an MQTT integration for one site from listening to traffic from another site. All MQTT payloads are in JSON format.

6.11.4 Creating an MQTT Integration

To connect to the MQTT feed, an Integration entry is required. This Integration can be created using the RESTful API or directly in the ICS web application.

6.12 API

Integrations can be scoped to an entire Organization, or for a specific site.

```
ORG Scope:

POST: https://test-security-
api.inovonics.com/v1/organizations/{{org_id}}/integration/

BODY: {
  "type": "MQTT",
  "additional_fields": {
    "mqtt_username": "exampleusername",
    "mqtt_password": "********"
  }
}
```

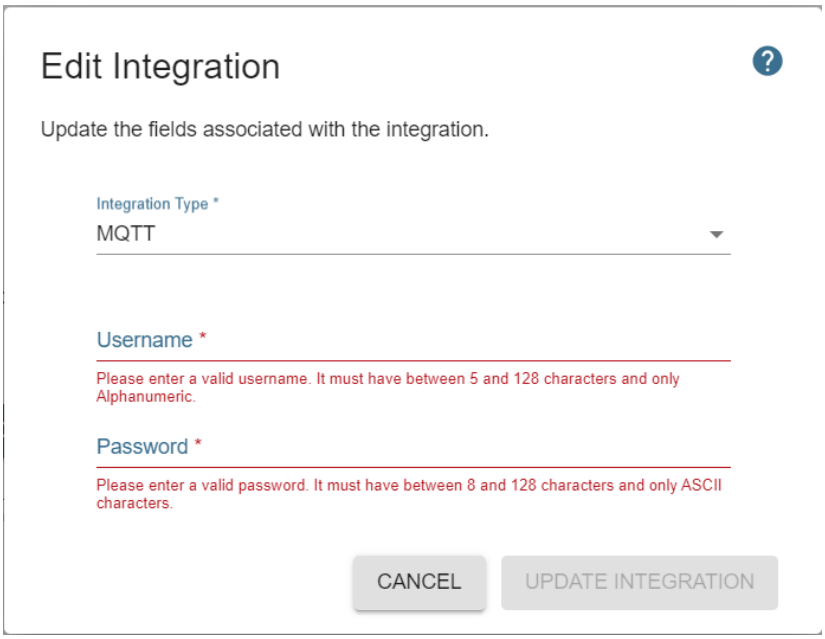
```
Site Scope:

POST: https://test-security-api.inovonics.com/v1/sites/{{site_id}}/integration/

BODY: {
  "type": "MQTT",
  "additional_fields": {
    "mqtt_username": "exampleusername",
    "mqtt_password": "********"
  }
}
```

Key Concepts

- *Organization scoped Integrations have access to MQTT messages for all Sites for the Organization.*
- *Site scoped Integrations have access to MQTT messages for only a singular Site.*

6.13 ICS

Configuration > Site Setup > Integration



Inputs:

- *Integration Type:* Please select “MQTT”
- *Username:* User provided value to be used by the MQTT client application to establish a connection. This value must be unique, and we highly recommend a GUID value. (5-128 alphanumeric characters)
- *Password:* User provided value to be used by the MQTT client application to establish a connection. We highly recommend a GUID value. (8-128 ASCII characters)

Once the integration is created, the connected application will be able to subscribe to the MQTT feed.

Details on the various Topics and message payloads can be found in the MQTT YAML documentation.

7 Feature Guide

7.1 Advanced Location

This section is intended to provide high level guidance for integrating your application with the Advanced Location features of the Inovonics Cloud Services Application. This guide assumes the reader is already familiar with ICS APIs and the structure and hierarchy of the application.

Advanced Location supports multiple Roaming Device types, including:

- Senior Living Pendants
- Mobile Duress Devices
- Asset Tags & Staff Badges

Advanced Location supports these features for the various Roaming Devices:

- Find Roaming Device
- Geofencing

Key Concepts

- *All location APIs will return the last available location for a Roaming Device. It is possible that this location could become stale if a pendant is not in range of a locator. If you consistently get stale or outdated locations, please ensure that your pendant's batteries are regularly changed, and that your locator network is expansive enough to keep pendants in range.*
- *It is possible for every location API to return two locations for one pendant. This is considered a "contested location" and occurs when two Inovonics locators both reported a similar signal strength. If this occurs, you will receive two locations in the payload, per the example below.*
- *Currently the location APIs will only search the last 24 hours of location history. If a pendant has not been in range of a locator for 24 hours, a location API call will not return a location for this pendant.*
- *Currently supports these pendant models that utilize BLE technology (Bluetooth Low Energy):*
  - *EN2221S-60*
  - *EN2222S-60*

7.1.1 Find Roaming Device

Retrieve a Roaming Device's last known location.

7.1.1.1 Request (GET)

`/v1/sites/{site_id}/transmitter/{transmitter_id}/location/`

7.1.1.2 Request Parameters

- `{site_id}`: ID of the site (required)
- `{transmitter_id}`: ID of the specific Roaming Device (required)

7.1.1.3 Response (Single Location)


```
{
    "location": {
         "locator_id": "43072f72-9ac6-ffff-a7c7-202ae2e341f1",
         "locator_name": "Room #104",
         "locator_txid": 2614999,
         "rssi": -29,
         "timestamp": "Tue, 05 Sep 2023 15:15:24 GMT"
    },
    "model": "EN2222S60",
    "name": "Jonathan F's Pendant,
    "transmitter_id": "e859fc72-ffff-45c2-b8f6-0220a8b6779f",
    "txid": 5589018
}
```


This shows the timestamp and locator info of the last time that the Roaming Device checked in with the ICS application. It also includes the Roaming Device's info itself.

7.1.1.4 Response (Contested Location)


```
{
"locations": [
   {
   "contested_location": true,
       "locations": [
           {
               "locator_id": "9d00941d-8744-4b10-ffff-44a546e3ca20",
               "locator_name": "Outside Corridor",
               "locator_txid": 25773211,
               "rssi": -43,
               "timestamp": "Tue, 05 Sep 2023 16:24:18 GMT"
           },
           {
               "locator_id": "9f905023-ffff-45fe-8eee-7d93b81fb7fd",
               "locator_name": "West Cafeteria",
               "locator_txid": 2573654,
               "rssi": -46,
               "timestamp": "Tue, 05 Sep 2023 16:24:18 GMT"
           }]
   }],
   "model": "EN2221S60",
   "name": Arnold C.,
   "transmitter_id": "e7df02b2-ffff-40f8-9564-1ed352ce40a3",
   "txid": 14026174
}
```


Please Note

- *A boolean variable called "contested_location" to denote the contested location.*
- *The locations are represented in an array containing two location objects.*
- *The Roaming Device information is the same JSON object.*

7.1.2 Find Roaming Device with Multiple Locations

Retrieve multiple locations for a single Roaming Device.

7.1.2.1 Request (GET)

```
/v1/sites/{site_id}/transmitter/{transmitter_id}/location/history/
?checkins=2
?{start_date}= 2022-01-01T12:00:00.000000-07:00
?{end_date}= 2022-01-01T15:00:00.000000-07:00?
```

7.1.2.2 Request Parameters

- `{site_id}`: ID of the site (required)
- `{transmitter_id}`: ID of the specific Roaming Device (required)
- `{checkins}`: The number of previous locations to fetch (required, between 1-120)
- `{start_date}`: The start date & time for locations to fetch (optional, ISO8601 datetime format)
- `{end_date}`: The end date & time for locations to fetch (optional, ISO8601 datetime format)

7.1.2.3 Response (Multiple Locations)


```
{
    "locations": [
        {
            "locator_id": "a9585947-f94f-bbbb-abad-813dc79bb295",
            "locator_name": "Newsom Courtyard",
            "locator_txid": 3239845,
            "rssi": -57,
            "timestamp": "Tue, 05 Sep 2023 17:08:10 GMT"
        },
        {
            "locator_id": "49aaaa41-726d-4e77-baf7-81a8d946c2c6",
            "locator_name": "Game Room",
            "locator_txid": 3243591,
            "rssi": -36,
            "timestamp": "Tue, 05 Sep 2023 17:07:24 GMT"
        }
    ],
    "model": "EN2222S60",
    "name": "Patient Zero",
    "transmitter_id": "e859fc72-8c8e-9999-b8f6-0220a8b6779f",
    "txid": 553461
}
```


Please Note

- *The same as the contested location response without the "contested_location" boolean variable.*
- *The locations are represented in an array of location objects.*
- *The Roaming Device information is the same JSON object.*

7.1.3 Find All Roaming Device's Location

Retrieves the last known location for all Roaming Devices for the given site.

7.1.3.1 Request (GET)

/v1/sites/{site_id}/transmitters/location

7.1.3.2 Request Parameters

- *{site_id}*: ID of the site (required)

7.1.3.3 Response


```
[
    {
        "location": {
            "contested_location": true,
            "locations": [
                {
                    "locator_id": "ac8674a6-abcd-40ab-96ba-d31f331bec4e",
                    "locator_name": "Wellness Room",
                    "locator_txid": 2579635,
                    "rssi": -55,
                    "timestamp": "Tue, 05 Sep 2023 21:02:53 GMT"
                },
                {
                    "locator_id": "1111115-dee1-4935-88d3-69bccf907f51",
                    "locator_name": "Purgatory",
                    "locator_txid": 5570777,
                    "rssi": -56,
                    "timestamp": "Tue, 05 Sep 2023 21:02:53 GMT"
                }
            ]
        },
        "model": "EN2222S60",
        "name": "Dulles",
        "transmitter_id": "19265c6-20e1-483e-8ee6-ae2a72be7f34",
        "txid": 324456
    },
    {
        "location": {
            "locator_id": "6beb4444-f8dd-445b-9ee2-1212178114a7",
            "locator_name": "Garage",
            "locator_txid": 3246387,
            "rssi": -51,
            "timestamp": "Tue, 05 Sep 2023 21:03:22 GMT"
        },
        "model": "EN2221S60",
        "name": "Willards",
        "transmitter_id": "3abcf2-ce42-47ae-b781-1ef87fd4f489",
        "txid": 5496262
    }
```


Please Note

- *The response will contain an array of Roaming Device and location objects.*

- *The response may contain a contested location response for a Roaming Device's location.*

7.1.4 Find All Roaming Devices and Multiple Locations

Retrieves multiple locations for all Roaming Devices for a given site.

7.1.4.1 Request (GET)

```
/v1/sites/{site_id}/transmitter/location/history/?checkins=2
?{start_date}= 2022-01-01T12:00:00.000000-07:00
?{end_date}= 2022-01-01T15:00:00.000000-07:00?
```

7.1.4.2 Request Parameters

- `{site_id}`: ID of the site (required)
- `{checkins}`: The number of previous locations to fetch (required, between 1-120)
- `{start_date}`: The start date & time for locations to fetch (optional, ISO8601 datetime format)
- `{end_date}`: The end date & time for locations to fetch (optional, ISO8601 datetime format)

7.1.4.3 Response


```
[
    {
        "locations": [
            {
                "locator_id": "7a9809c1-0fb0-4e89-a367-74ae32c78a8a",
                "locator_name": "Lab",
                "locator_txid": 2614083,
                "rssi": -59,
                "timestamp": "Tue, 05 Sep 2023 21:14:57 GMT"
            },
            {
                "locator_id": "7a9809c1-0fb0-4e89-a367-74ae32c78a8a",
                "locator_name": "Yoga Room",
                "locator_txid": 2614980,
                "rssi": -60,
                "timestamp": "Tue, 05 Sep 2023 21:14:28 GMT"
            }
        ],
        "model": "EN2221S60",
        "name": "Gallup E.",
        "transmitter_id": "76debe95-8e3a-4a08-9966-29cd172214bf",
        "txid": 10866411
    },
    {
        "locations": [
            {
                "locator_id": "2e4323b-3787-4533-9b40-9cc2e3bbb2ea",
                "locator_name": "PT Room",
                "locator_txid": 2576987,
                "rssi": -52,
                "timestamp": "Tue, 05 Sep 2023 21:14:55 GMT"
            },
            {
                "contested_location": true,
                "locations": [
                    {
                        "locator_id": "2e765413b-3787-4533-9b40-9cc2e3bbb2ea",
                        "locator_name": "PT Room",
                        "locator_txid": 2576821,
                        "rssi": -53,
                        "timestamp": "Tue, 05 Sep 2023 21:14:04 GMT"
                    },
                    {
                        "locator_id": "37d3d7fa-1234-4b8f-9edb-f2ca8767ae88",
                        "locator_name": "Phone Room",
                        "locator_txid": 1980334,
                        "rssi": -56,
                        "timestamp": "Tue, 05 Sep 2023 21:14:04 GMT"
                    }
                ]
            }
        ],
        "model": "EN2221S60",
        "name": "Alistair P",
        "transmitter_id": "81e7f433-6a85-4d29-6543-f8258ebd10ec",
        "txid": 1375606}
```

Please Note

- *The response will contain an array of Roaming Device and location objects.*
- *The response may contain a contested location response for a Roaming Device's location.*

7.1.5 Geofencing

Geofencing is a means of restricting access to specific areas within a Site. Geofencing settings can be managed in the ICS UI through the Advanced Location

Key Concepts

- *Geofencing restrictions are built around the concept of a Unit's Schedule.*
- *Schedules denote the days and times when a Unit is 'Open'.*
- *Schedules can support multiple 'Open' time slots on a single day.*
- *Schedules can have no 'Open' days and times to denote the Unit is always Restricted.*
- *Schedules can apply to all Roaming Device or to specific devices.*
- *When a Roaming Device is detected in a restricted Unit, a 'Breach' event is triggered.*

7.1.5.1 Create / Update a Schedule

These schedules apply to the entire unit. A unit can only have one schedule. Any Roaming Device that accesses outside of permitted hours will generate a breach. If no time slot is specified, the schedule will be set as "never_open": true, which means that it is is blocked at any time of any day.

7.1.5.1.1 Request (POST / PUT)

```
/v1/sites/{site_id}/units/{unit_id}/schedule/
```

7.1.5.1.2 Request parameters

```
{site_id}: ID of the site (required)
{unit_id}: ID of the unit (required)
```

7.1.5.1.3 Request body

```
{name}: string indicating the name of the schedule
{note}: string with a note related to the schedule
{sunday}: list of timeslots {"start_time": "HH:mm", "end_time": "HH:mm"}
{monday}: list of Timeslots
{monday}: list of Timeslots
{wednesday}: list of Timeslots
{monday}: list of Timeslots
{friday}: list of Timeslots
{monday}: list of Timeslots

Timeslot model: {"start_time": "HH:mm", "end_time": "HH:mm"}
```

7.1.5.1.4 Response

```
{
       "schedule_id": "4012edd6-e0f8-419b-990b-86f3b06a06c2",
       "name": "Schedule for Office 1",
      "note": "Open in active hours from mon to thu",
      "never_open": false,
      "sunday": [],
      "monday": [
      {
       "end_time": "18:00",
            "start_time": "09:00"
       }],
      "tuesday": [
      {
       "end_time": "18:00",
       "start_time": "09:00"
       }],
       … entry for each day of the week and it's open time slots
```

7.1.5.2 Retrieve a Schedule

7.1.5.2.1 Request (GET)

```
/v1/sites/{site_id}/units/{unit_id}/schedule/
```

7.1.5.2.2 Request parameters

```
{site_id}: ID of the site (required)
{unit_id}: ID of the unit (required)
```

7.1.5.2.3 Response

```
{
    "schedule_id": "4012edd6-e0f8-419b-990b-86f3b06a06c2",
    "name": "Schedule for Office 1",
    "note": "Open in active hours from mon to thu",
    "never_open": false,
    "sunday": [],
    "monday": [
    {
    "end_time": "18:00",
          "start_time": "09:00"
    }],
    "tuesday": [
    {
    "end_time": "18:00",
          "start_time": "09:00"
    }],
    … entry for each day of the week and it's open time slots
```

7.1.6 Roaming Device Schedules

These schedules apply to a specific Roaming Device. A Roaming Device can only have one schedule per Unit. If the Roaming Device is detected in the restricted unit outside of 'open' hours then a breach event will be generated. If no 'open' time slot is specified, the schedule will be set as "never_open": true, meaning this Roaming Device is always restricted from this Unit.

7.1.6.1 Create / Update a Unit + Roaming Device Schedule

7.1.6.1.1 Request (POST / PUT)

/v1/sites/{site_id}/units/{unit_id}/transmitters/{transmitter_id}/schedule/

7.1.6.1.2 Request parameters

{site_id}: ID of the site (required)
{unit_id}: ID of the unit (required)
{transmitter_id}: ID of the Roaming Device (required)

7.1.6.1.3 Request body

{name}: string indicating the name of the schedule
{note}: string with a note related to the schedule
{sunday}: list of timeslots {"start_time": "HH:mm", "end_time": "HH:mm"}
{monday}: list of Timeslots
{monday}: list of Timeslots
{wednesday}: list of Timeslots
{monday}: list of Timeslots
{friday}: list of Timeslots
{monday}: list of Timeslots

Timeslot model: {"start_time": "HH:mm", "end_time": "HH:mm"}

7.1.6.1.4 Response

```
{
    "schedule_id": "4012edd6-e0f8-419b-990b-86f3b06a06c2",
    "name": "Schedule for Office 1",
    "note": "Open in active hours from mon to thu",
    "never_open": false,
    "sunday": [],
    "monday": [
    {
    "end_time": "18:00",
          "start_time": "09:00"
    }],
    "tuesday": [
    {
    "end_time": "18:00",
          "start_time": "09:00"
    }],
    … entry for each day of the week and it's open time slots
```

7.1.6.2 Retrieve a Unit + Roaming Device Schedule

7.1.6.2.1 Request (GET)

```
/v1/sites/{site_id}/units/{unit_id}/transmitters/{transmitter_id}/schedule/
```

7.1.6.2.2 Request parameters

```
{site_id}: ID of the site (required)
{unit_id}: ID of the unit (required)
{transmitter_id}: ID of the Roaming Device (required)
```

7.1.6.2.3 Response

```
{
   "schedule_id": "4012edd6-e0f8-419b-990b-86f3b06a06c2",
   "name": "Schedule for Office 1",
   "note": "Open in active hours from mon to thu",
   "never_open": false,
   "sunday": [],
   "monday": [
   {
   "end_time": "18:00",
         "start_time": "09:00"
   }],
   "tuesday": [
   {
   "end_time": "18:00",
         "start_time": "09:00"
   }],
   … entry for each day of the week and it's open time slots
```

7.1.6.3 Site Schedules

Retrieve all Schedules associated with a given site.

The response object is an array of Schedules as defined below.

7.1.6.3.1 Request (GET)

```
/v1/sites/{site_id}/schedules/
```

7.1.6.3.2 Request parameters

```
{site_id}: ID of the site (required)
```

7.1.6.3.3 Response

```
[
   {
       "building_id": "61bbbc80-5d76-4ea9-a148-d9cf6711d81f",
       "building_name": "San Serafin Hospital",
       "floor_id": "6e201637-328b-44b7-8dc3-92bc2bf63457",
       "floor_name": "Main Floor",
       "unit_id": "1cd2fd72-6098-4b25-8200-1e7078c9f74a",
       "unit_name": "Waiting Room",
       "pendant_id": "e6772f14-8d41-4ebd-8a71-2120708f0c76",
       "pendant_name": "asset tag ji",
       "pendant_txid": 9832146,
       "schedule_id": "f0c9d926-e29b-4360-b65d-6293a24bae5b",
       "name": "2",
       "never_open": false,
       "note": "",
       "sunday": [] …array of Timeslot objects,
       "monday": [] …array of Timeslot objects,
       "tuesday": [] …array of Timeslot objects,
       "wednesday": [] …array of Timeslot objects,
       "thursday": [] …array of Timeslot objects,
       "friday": [] …array of Timeslot objects,
       "saturday": [] …array of Timeslot objects,
   },
   { … another Schedule object}
]
```

7.1.7 Breaches

Breaches are triggered when a Roaming Device is detected in a unit outside of the ‘open’ hours defined in the Schedules. They are created automatically by ICS and cannot be created via the API.

7.1.7.1 Breach Types

Breach Types define the type of Geofence restriction that was broken. The four breach types are:

1. *"unit_breach": A Roaming Device is detected in a ‘never-open’ restricted unit.*
2. *"unit_schedule_breach": A Roaming Device is detected outside of the unit’s ‘open’ schedule.*
3. *"unit_transmitter_breach": The specific Roaming Device is detected in a ‘never-open’ restricted unit.*
4. *"unit_transmitter_schedule_breach": The specific Roaming Device is detected outside of the unit’s ‘open’ schedule.*

7.1.7.2 Breach Statuses

1. *“Initial”: denotes the first time this Roaming Device was detected in this restricted Unit.*
2. *“Ongoing”: denotes the Roaming Device is still detected in the restricted Unit.*
3. *“Clear”: denotes the Roaming Device has been detected in a non-restricted Unit.*

7.1.7.3 Retrieve Breaches for a Roaming Device (Last 24 hours)

7.1.7.3.1 Request (GET)

```
/v1/sites/{site_id}/transmitters/{transmitter_id}/breaches/
```

7.1.7.3.2 Request parameters

```
{site_id}: ID of the site (required)
{transmitter_id}: ID of the pendant (required)
```

7.1.7.3.3 Response


```
{
    "breach_id": "ac73e3c0-dc33-4423-0443-58adbda89dc4",
    "timestamp": "2024-03-11T17:00:05.062Z",
    "status": "Initial",
    "breach_types": "unit_breach",
    "last_modified_time": "2024-03-11T17:00:05.062Z",
    "clear_time": "2024-03-11T17:00:05.062Z",
    "unit_name": "Unit A",
    "clear_unit_name": "Unit A",
    "site_name": "Site A",
    "device_name": "Device A",
    "clear_unit_id": "d8a6ac32-a484-ed19-c3fb-231412ffd93f",
    "unit_id": "e9c51455-4b56-2701-4f3e-2a36ac92421c",
    "initial_locator_id": "9ad59d4b-84ec-1c30-843f-51d6eadf0900",
    "initial_rssi": -50
}
```


7.1.7.4 Retrieve a Breach

Retrieve the details for a specific Breach event.

7.1.7.4.1 Request (GET)

```
/v1/sites/{site_id}/breaches/{breach_id}/
```

7.1.7.4.2 Request parameters

```
{site_id}: ID of the site (required)
{breach_id}: ID of the breach (required)
```

7.1.7.4.3 Response


```
{
    "breach_id": "ac73e3c0-dc33-4423-0443-58adbda89dc4",
    "timestamp": "2024-03-11T17:00:05.062Z",
    "status": "Initial",
    "breach_types": "unit_breach",
    "last_modified_time": "2024-03-11T17:00:05.062Z",
    "clear_time": "2024-03-11T17:00:05.062Z",
    "unit_name": "Unit A",
    "clear_unit_name": "Unit A",
    "site_name": "Site A",
    "device_name": "Device A",
    "clear_unit_id": "d8a6ac32-a484-ed19-c3fb-231412ffd93f",
    "unit_id": "e9c51455-4b56-2701-4f3e-2a36ac92421c",
    "initial_locator_id": "9ad59d4b-84ec-1c30-843f-51d6eadf0900",
    "initial_rssi": -50
}
```


7.1.8 Breach Comments

Breach events support adding Comments to them for recording any relevant details.

7.1.8.1 Create a Breach Comment

7.1.8.1.1 Request (POST)

```
/v1/sites/{site_id}/breaches/{breach_id}/comments/
```

7.1.8.1.2 Request parameters

```
{site_id}: ID of the site (required)
{breach_id}: ID of the breach (required)
```

7.1.8.1.3 Request body

```
{
    "comment": "sample comment"
}
```

7.1.8.1.4 Response

```
[{
    "username": "someone@example.com",
    "timestamp": "2024-03-11T17:07:43.027Z",
    "alarm_id": "72793372-ba2a-a295-6e22-b5f8fff6221c",
    "comment_id": "61f7b5eb-c925-0fe6-cce8-e640e7de783b",
    "comment": "sample comment"
}]
```

Please Note: the response is an array of Note objects.

7.1.8.2 Retrieve Comments for a Breach

7.1.8.2.1 Request (GET)

```
/v1/sites/{site_id}/breaches/{breach_id}/comments/
```

7.1.8.2.2 Request parameters

```
{site_id}: ID of the site (required)
{breach_id}: ID of the breach (required)
```

7.1.8.2.3 Response

```
[
    {
        "username": "someone@example.com",
        "timestamp": "2024-03-11T17:07:43.027Z",
        "alarm_id": "72793372-ba2a-a295-6e22-b5f8fff6221c",
        "comment_id": "61f7b5eb-c925-0fe6-cce8-e640e7de783b",
        "comment": "Test comment"
    },
    {…another comment object}
]
```

7.2 Fall Detection for Senior Living

The Inovonics Fall Detection solution allows you to easily upgrade your trusted e-call system to include a critical alarm feature for high-risk events and data insights that can help you to improve resident outcomes.

7.2.1 Overview

Fall Detection utilizes multiple sensors on the pendant worn by senior living residents, coupled with the vast computing power of the cloud, to provide valuable information about resident falls. Additionally, ICS provides reporting & analytics for fall data.

7.2.2 Fall Detection Mode

The Inovonics Fall Detection pendant supports 2 modes of operation:

- *"Fall Mode": The pendant utilizes its advanced algorithms for fall detection and sends alarms through EchoStream that are classified as 'Falls'.* ***This is the default mode of the pendant.***

- *"Activity Mode": The pendant utilizes its advanced algorithms for fall detection and sends alarms through EchoStream that are classified as 'Activity'. Please refer to our installation and operation manual for detailed instructions.*

7.2.3 Retrieve Fall Detection Mode (API)

The current mode of Fall Detection pendants can be retrieved from the API for either an individual pendant, or for all pendants registered to a given site.

7.2.3.1 Request (GET)

```
/v1/sites/{site_id}/transmitters/{transmitter_id}
```

7.2.3.2 Request Parameters

- `{site_id}`: ID of the site (required)
- `{transmitter_id}`: ID of a specific transmitter / pendant (optional)

7.2.3.3 Response


```
{
    "alarm_category": "Life Safety",
    "bleid": 0,
    "device_profile": {
            "additional_fields": {
                    "mode": "activity"
                    },
            "model": "EN2222S60",
            "name": "default_en2222s60_profile_activity_monitor_mode",
            "note": "Default EN2222S60 Profile defined by system and cannot be modified",
            "system_defined": true
            },
    "device_profile_id": "1931e224-c9dd-4928-aa3f-b2005c03d8f7",
    "device_type": "Pendant",
    "favorite": true,
    "first_cloud_checkin_date": "2024-03-19 18:28:18.838992",
    "last_low_battery_end_time": "None",
    "last_low_battery_start_time": "None",
    "last_replacement_battery_end_time": "None",
    "last_replacement_battery_start_time": "None",
    "last_sync_date": "2024-03-21T16:21:55.638560-06:00",
    "location": "Example pendant",
    "model": "EN2222S60",
    "note": "",
    "test_device": false,
    "transmitter_id": "3e4654e0-2fe5-49e7-aff0-93432fce28b6",
    "txid": 3246682
}
```

The supported “mode” values are:

- *"alarm": Fall Detection is ON (default)*
- *"activity": Fall Detection is OFF*

7.2.4 Retrieve Fall Detection Mode (MQTT)

Fall Detection Mode changes can also be detected in real-time via the MQTT feed.

7.2.4.1 Topic

```
<username>/site/<site_id>/devices
```

7.2.4.2 Example (Disabled)


```
{
  "organization_name": "Senior Living System Test Sites",
  "site_name": "Example Site",
  "building_name": null,
  "device_location": "Example pendant",
  "device_note": "",
  "device_txid": 3233220,
  "device_type": "TRANSMITTER",
  "device_model": "EN2222S60",
  "last_sync_time": "2024-03-21T22:49:39.583401",
  "send_time": "2024-03-21T22:49:42.053589",
  "status": "FALL_MODE_DISABLED",
  "rfmessage_id": "2d474e0c-1c7c-4989-9e1e-a10ea2e83aab",
  "is_historical": false,
  "rfmessage_time": "2024-03-21T22:49:39.583401"
}
```


7.2.4.3 Example (Enabled)


```
{
  "organization_name": "Senior Living System Test Sites",
  "site_name": "Example Site",
  "building_name": null,
  "device_location": "Example pendant",
  "device_note": "",
  "device_txid": 3233220,
  "device_type": "TRANSMITTER",
  "device_model": "EN2222S60",
  "last_sync_time": "2024-03-21T22:49:39.583401",
  "send_time": "2024-03-21T22:49:42.053589",
  "status": "FALL_MODE_DISABLED",
  "rfmessage_id": "2d474e0c-1c7c-4989-9e1e-a10ea2e83aab",
  "is_historical": false,
  "rfmessage_time": "2024-03-21T22:49:39.583401"
}
```

8 Embedded Widgets

8.1 Overview

To facilitate ease of feature integration, Inovonics Cloud Services now supports Embedded Widgets. These widgets encapsulate both user interface & feature logic that can be included within partner applications.

Key Concepts

- *Embedded Widgets are delivered via specialized URLs from Inovonics Cloud Services and should be implemented inside of an <iframe> object in your application.*
- *User access to any Embedded Widget within your application should be determined by your application's user role & permissions system.*

8.2 Available Widgets

The Inovonics Cloud Service API currently supports these widgets:

8.2.1 Find Roaming Device

Find Roaming Device is part of the Advanced Location solution. It supports:

- Roaming Device Search
- Floor map location
- Date & time of last pendant check-in
- Roaming Device Search History
- "Starred" Roaming Device for quick access

All of this functionality is encapsulated in this single powerful widget.

8.2.2 Operational Insights Dashboards

Operational Insights Dashboards offer powerful reporting tools for your customers.

There are 4 distinct dashboards available:

- Alarms Dashboard
- Battery Dashboard
- Fall Alarms Dashboard (Available for Senior Living customers only)
- Fall History Dashboard (Available for Senior Living customers only)

8.3 Authorization

Embedded Widgets supports the bearer authentication scheme. The flow follows this sequence:

1. Request an *{access_token}* bearer token.
2. Utilize that *{access_token}* to retrieve a widget's URL.

8.3.1 Request

Retrieve an *{access_token}* that will be able to retrieve a widget's URL.

https://security-api.inovonics.com/oauth/token/

Request Header

Content-Type application/x-www-form-urlencoded

8.3.2 Request Parameters

- *{grant_type}*: The string *"password"* is the only accepted value (required)
- *{client_id}*: client identifier (required)
- *{username}*: the username, typically an email address (required)
- *{password}*: the username's password (required)
- *{client_secret}*: client secret (required)

8.3.3 Response

```
{
   access_token: "sDaPb7KmgpVNDdmFuviB2i4RzllL3EwxQL047MrSJC",
   expires_in: 86400,
   refresh_token: "98h3sBoj03abGMYKDsWefCd3eMDac51IchWDp9NBZcT4rVhU",
   scope: "webapp",
   token_type: "Bearer",
   user_id: "5bg935b6-k5l2-94os-b204-923peof2nl40",
   username: "example@example.com",
}
```

8.3.4 Response Fields

The response will contain these fields which will be utilized when requesting a widget:

- *{access_token}* (string): A bearer token that grants secure access API resources.
- *{expires_in}* (number): The lifespan of the bearer token in milliseconds.
- *{refresh_token}* (string): Token that enables reauthorization without requiring a new login.
- *{scope}* (string): The context which the token may access.
- *{token_type}* (string): The type of token. Supports "bearer".
- *{user_id}* (string): The unique UUID of the requesting user.
- *{username}* (string): the username, typically an email address.

8.4 Requesting a Widget

Using the Auth `{access_token}`, a subsequent API can call be made to fetch the widget's URL

8.4.1 Request Structure

All Widget requests follow this structure:

`/v1/sites/{site_id}/{widget}/?scope={scope}&org_id={organization_id}`

8.4.2 Request Parameters & Headers

- `{site_id}`: ID of the site (required)
- `{organization_id}`: ID of the organization (required)
- `{scope}`: context that the bearer token may access (required)
- `{widget}`: the widget being requested (required)

Embedded Widget requests requires an authorization header that includes the bearer token

```
'Authorization': 'Bearer ' + {access_token}
```

8.4.2.1 Scope

The `{scope}` identifies the context that the bearer token may access.

Available scopes:

- `"location_find"` grants access to the Find Roaming Device widget.
- `"site_dashboard"` grants access to the various Operational Insights Dashboards.

8.4.2.2 Widget

The `{widget}` identifies the specific widget that is being requested.

Available Features:

- `"embedded-find-them-url"`: Find Roaming Device widget's.
- `"embedded-alarms-dashboard-url"`: Operational Insights – Alarms Dashboard.
- `"embedded-battery-dashboard-url"`: Operational Insights – Battery Dashboard.
- `"embedded-fall-alarms-dashboard-url"`: Operational Insights – Fall Alarms Dashboard.
- `"embedded-fall-alarms-history-dashboard-url"`: Operational Insights – Fall Alarms History Dashboard.

8.4.2.3 Scope & Widget compatibility table

Scope	Widget
location_find	*embedded-find-them-url*
site_dashboard	*embedded-alarms-dashboard-url*
site_dashboard	*embedded-battery-dashboard-url*
site_dashboard	*embedded-fall-alarms-dashboard-url*
site_dashboard	*embedded-fall-alarms-history-dashboard-url*

8.4.3 Request Examples

The Scope parameter is highlighted in green; the Widget in teal.

Find Roaming Device

*/v1/sites/123asd456/**embedded-find-them-url**/?scope=**location_find**&org_id=890qwe678*

Operational Insights: Alarms Dashboard

*/v1/sites/123asd456/**embedded-alarms-dashboard-url**/?scope=**site_dashboard***
&org_id=890qwe678

Operational Insights: Battery Dashboard

*/v1/sites/123asd456/**embedded-battery-dashboard-url**/?scope=**site_dashboard***
&org_id=890qwe678

8.4.4 Response Example

The successful response will contain the URL for the requested widget. Here is an example of a the "Find Roaming Device" widget response.

```
{
   "url": "https://security.inovonics.com/security/282w8w8e92/sites/29e9e9-8e8fe9/find-them"
}
```

8.5 Embedding the Widget

8.5.1 IFrame Settings & Styles

As stated earlier, widgets are intended to be embedded within an *iframe* control.

To ensure proper behavior, Inovonics suggests the *iframe* control have these settings & values set:

Setting	Value	Description
allowRedirectInIframe	True	Required for the proper function of the widget
sandbox	"allow-same-origin allow-scripts"	Enables proper restrictions for embedded content
src	Response URL	This is The URL returned from the widget request
frameborder	"0" (zero)	To display seamlessly within the embedding app.

Example

```
<iframe
    allowRedirectInIframe="true"
    sandbox="allow-same-origin allow-scripts"
    src=<response URL>
    frameborder="0"
></iframe>
```

Inovonics recommends explicitly setting the *iframe* style to ensure the optimal user experience. Each implementation may vary, and these are just general guidelines.

```
element {
  position: relative;
  height: 100%;
  width: 100%;
}
```

With these settings, the *iframe* would utilize the entire container's area, and should not affect the position of other UI elements.

Read More information about *iframe* settings.

9 Appendix

Below you will find other helpful resources for your integrations.

9.1 Connectivity Loss

There are several different scenarios for loss of connectivity, and Inovonics Cloud Services has monitoring and notifications in the event of connectivity loss. Below describe various outage scenarios and how the system will behave.

Element	Frequency	What Will Happen	Partner Action	Scope	Restoral
Google Cloud Platform	30 seconds (heartbeat)	MQTT "Loss of connection" message is sent	Subscribe to MQTT topic (see YAML docs)	Global	All available queued notifications are delivered.
Inovonics Cloud Services (ICS)	30 seconds (heartbeat)	Online messages will cease	Recommend monitoring for 2 concurrent missed heartbeats.	Global	All available queued notifications are delivered.
MQTT Broker	30 seconds (heartbeat)	Online messages will cease	Recommend monitoring for 2 concurrent missed heartbeats.	Global	All available queued notifications are delivered.
Gateway - Loss of Power	7.5 - 12.5 minutes	MQTT “Inactive" message is sent	Subscribe to MQTT topic (see YAML docs) *** Recommend a backup power source*	Local	New alarms will be delivered, however alarms that occurred during loss of power cannot be recovered.
Gateway - Connection to ICS lost	7.5 - 12.5 minutes	MQTT "Inactive" message is sent	Subscribe to MQTT topic (see YAML docs)	Local	All devices that alarmed during the outage will send alarms when the connection is restored
Gateway - Network connection lost	7.5 - 12.5 minutes	MQTT "Inactive" message is sent	Subscribe to MQTT topic (see YAML docs) *** Recommend a backup internet / cellular connection*	Local	All devices that alarmed during the outage will send alarms when the connection is restored

9.1.1 Message Examples

9.1.1.1 Loss of Connection

```
{
  "timestamp": "2022-10-05T23:52:49.376358",
  "status": "OFFLINE",
  "type": "Cloud Services Status",
  "description": "The following critical cloud services are down: <SERVICE 1> |
<SERVICE 2> | ... "
}
```

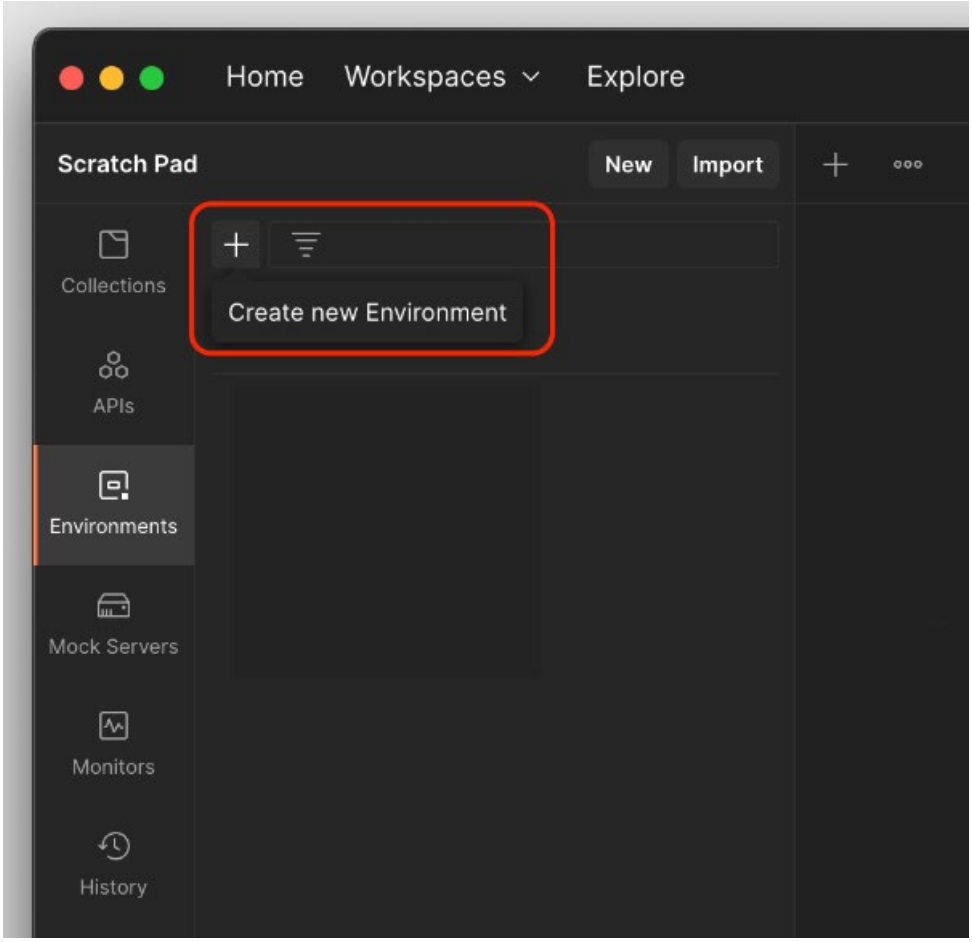
9.1.1.2 Inactive

```
{
    "organization_name": "Organization Name",
    "site_name": "Site Name",
    "building_name": "Building Name",
    "device_location": "Device Location",
    "device_txid": 123456,
    "device_type": "GATEWAY",
    "device_model": "EN7380",
    "last_sync_time": "2023-07-18T12:42:46.593641",
    "send_time": "2023-07-18T12:53:45.783197",
    "status": "INACTIVE",
    "rfmessage_id": ""
}
```

9.2 Postman

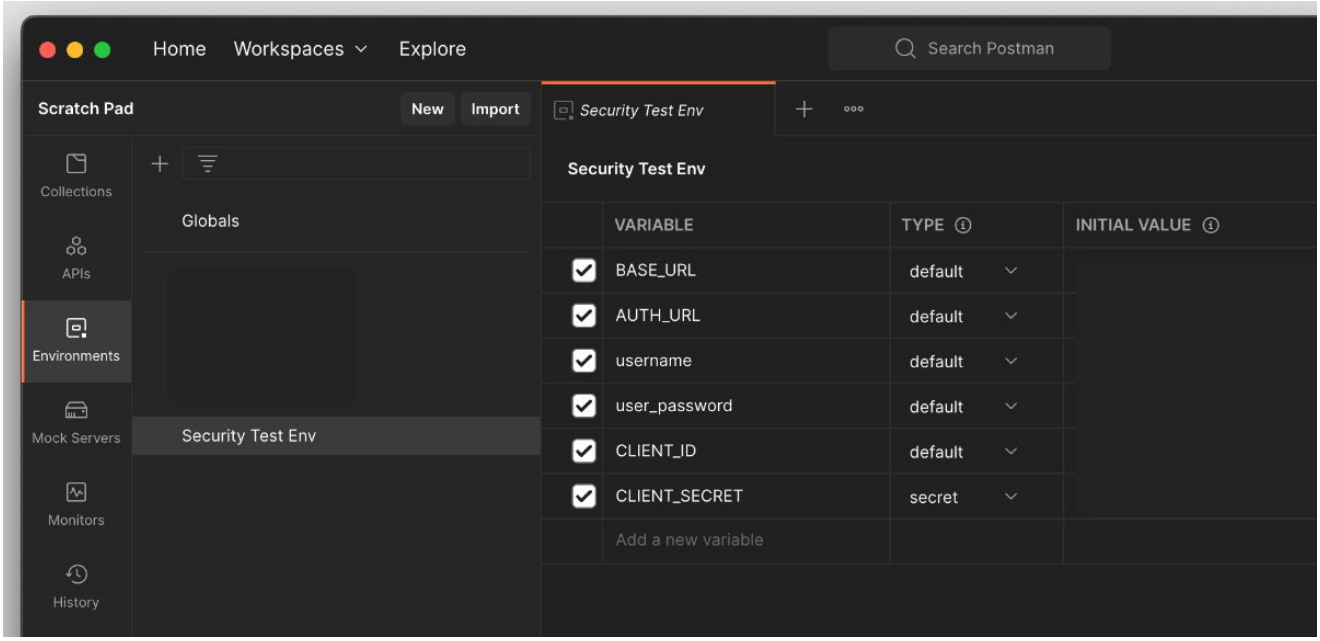
Postman is a tool that can be used to rapidly consume API services and prototype integration.

Open Postman > go to "*Environment*" tab > click on "+" (Create new Environment).



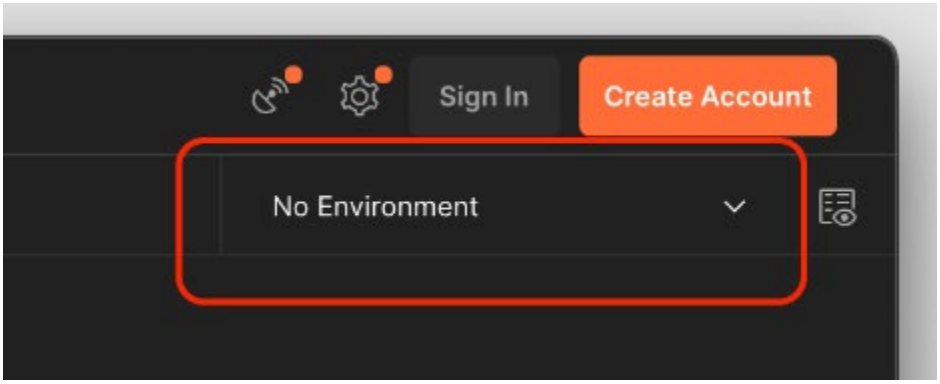
1. Name the New Environment (ex: Inovonics Test Env).
2. Add the following variables.
3. Assign their initial values.
4. Save your changes.

Variable	Initial Value
BASE_URL	https://test-security-api.inovonics.com
AUTH_URL	https://test-security-api.inovonics.com/oauth/token/
username	your username
user_password	your password
CLIENT_ID	Your client id
CLIENT_SECRET	Your client secret



Be sure to click on Reset All if the values are not reflected in the Current Value column.

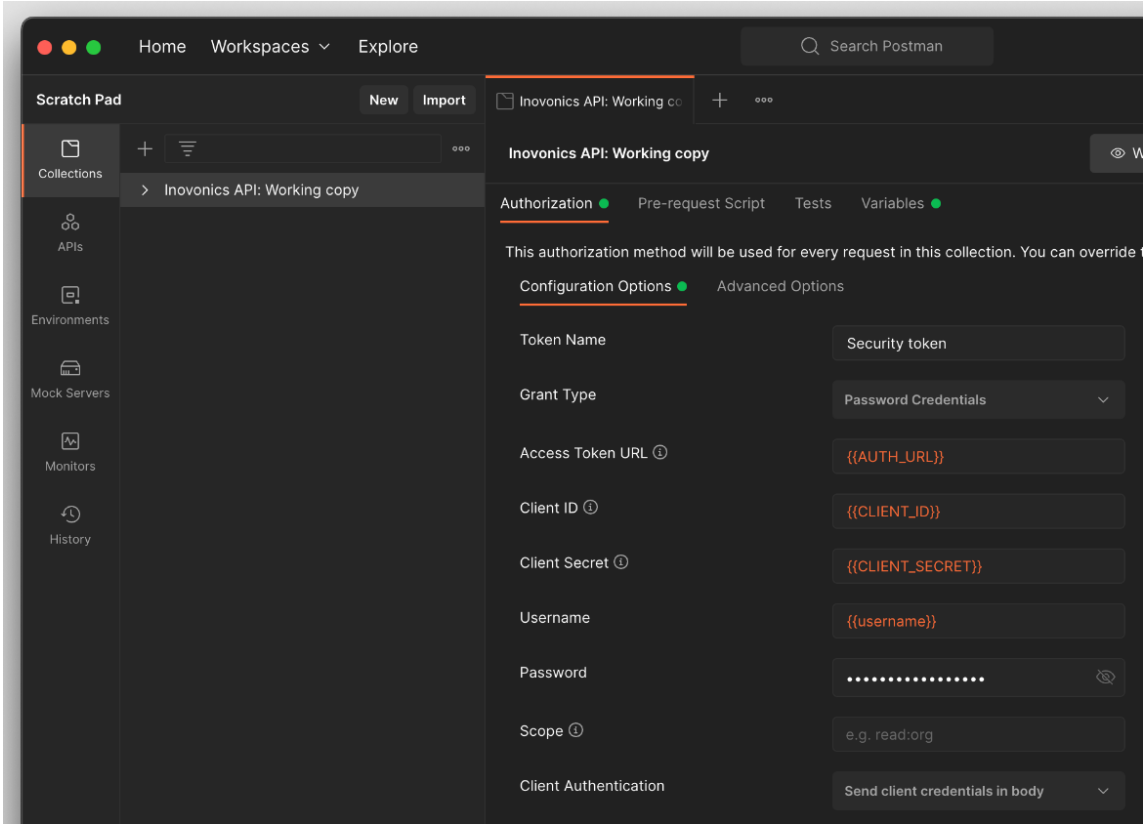
On the top right corner of the Postman, click on the dropdown section and select this new environment that was created.



Click on “*Collections*” tab > go to your collection’s root folder >Click on Authorization Tab

Scroll to the “*Configure New Token*” section and enter the following details:

Field	Value
Token Name	any name
Grant Type	Password Credentials
Access Token URL	{{AUTH_URL}}
Client ID	{{CLIENT_ID}}
Client Secret	{{CLIENT_SECRET}}
Username	{{username}}
Password	{{user_password}}
Client Authentication	Send client credentials in body



Click on “*Get New Access Token*” this will generate a new token > once successful > proceed and click "*Use Token*”.

Now the access token is applied to the collection and API requests can be made.

In the individual API request tabs > under Authorization sub-tab > make sure the Type is “*Inherit auth from parent*”.

9.3 Inovonics Hardware Part Numbers

Hardware Component	Part Numbers	Application		
		Security	Senior Living	TapWatch
Gateway	EN7295	§		
	EN7380		§	
	EN7580			§
Repeater	EN5040	§	§	§
	EN5040-T	§	§	§
	EN5040-20T		§	
Locator	EN5060	§	§	
BLE-Enabled Pendants	EN2221S-60		§	
	EN2222S-60		§	
	EN2224	§		
	EN2233D	§		
	EN2233S	§		
	EN2235D	§		
	EN2235S	§		
	EN2236D	§		
	EN2238D	§		
BLE Asset Tags	ADVL-1280	§	§	
BLE Staff Badges	ADVL-1290	§	§	
	ADVL-2291	§	§	

9.4 MQTT YAML file content

The following YAML content may be rendered in your preferred YAML viewer, such as https://studio.asyncapi.com/.

```yaml
asyncapi: 2.5.0
info:
  title: Mobile Duress MQTT Location Integration
  version: 1.18.0.0
  description: >-
    Receive location messages for alarms taking place on sites via the Inovonics
    Cloud.
servers:
  production:
    url: mqtt.inovonics.com
    protocol: mqtt
    description: Production broker
    variables:
      port:
        description: Secure connection (TLS) is available through port 8883.
        default: '1883'
        enum:
          - '1883'
          - '8883'
  test:
    url: test-mqtt.inovonics.com
    protocol: mqtt
    description: Test broker
    variables:
      port:
        description: >-
          Secure connection (TLS) is unavailable on this URL, see
          test-alternative.
        default: '15856'
        enum:
          - '15856'
  test-alternative:
    url: m9cdfcd9.us-east-1.emqx.cloud
    protocol: mqtt
    description: Test broker
    variables:
      port:
        description: >-
          This broker is the same broker as test-mqtt.inovonics.com, just uses
          the broker base username.  Secure connection (TLS) is available
          through port 15811.
        default: '15856'
        enum:
          - '15856'
          - '15811'
defaultContentType: application/json
channels:
  '{username}/site/{site_id}/location':
    description: >-
      The topic on which location messages for a specific site are reported. It
      is highly recommended to subscribe to this topic using a QOS of 1 or 2,
      and using a persistent session (setting the clean session flag to false).
    parameters:
```

```
      site_id:
        $ref: '#/components/parameters/site_id'
      username:
        $ref: '#/components/parameters/username'
    subscribe:
      summary: Event describing a location for an alarm event on a site.
      message:
        $ref: '#/components/messages/location'
  '{username}/site/{site_id}/devices':
    description: >-
      The topic on which problematic device status messages (such as inactivity)
      for a specific site are reported. It is highly recommended to subscribe to
      this topic using a QOS of 1 or 2, and using a persistent session (setting
      the clean session flag to false).
    parameters:
      site_id:
        $ref: '#/components/parameters/site_id'
      username:
        $ref: '#/components/parameters/username'
    subscribe:
      summary: Event describing device problems on the site.
      message:
        $ref: '#/components/messages/device_status'
  status:
    description: The topic on which the health of the Inovonics Cloud can be monitored.
    subscribe:
      summary: Event indicating a health/status change of the Inovonics Cloud.
      message:
        $ref: '#/components/messages/system_health'
components:
  messages:
    system_health:
      summary: >-
        The current state of the Inovonics Cloud system health.  The system
        health will be sent once every 30 seconds to indicate whether all
        critical cloud services are functioning correctly.  This status does not
        take IP Gateway health into account, use the device_status topic to
        monitor gateways instead.
      payload:
        type: object
        properties:
          timestamp:
            description: An ISO8601 string in UTC at which the status change occurred.
            type: string
          type:
            description: >-
              This should always be 'Cloud Services Status' to indicate what
              this status is for (not for gateways).
            type: string
          status:
            description: Current state of the Inovonics Cloud system
            type: string
            enum:
              - ONLINE
              - OFFLINE
          description:
            description: A description of which specific cloud services are offline.
            type: string
    location:
      summary: >-
        Location message with the location information for an alarming pendant
        on a site
```

```
payload:
  type: object
  properties:
    status:
      description: >-
        The status of determining the location.  For each alarm, two
        messages will be published.  One indicating the alarm was received
        by the Inovonics cloud and the location is being calculated, and a
        second message once the location is determined or the cloud is
        unable to determine the location
      type: string
      enum:
        - CALCULATING
        - LOCATION_DETERMINED
        - LOCATION_CONTESTED
        - LOCATION_UNKNOWN
    send_time:
      description: >-
        An ISO8601 string in UTC at which location information was
        published.
      type: string
    pendant_txid:
      description: >-
        TXID (transmitter ID) of the pendant that sent an alarm on the
        site.
      type: integer
    pendant_name:
      description: >-
        The name of the pendant configured in the Inovonics Mobile Duress
        application.
      type: string
    organization_name:
      description: The name of the organization to which the site belongs.
      type: string
    site_name:
      description: The name of the site to which the alarming pendant belongs.
      type: string
    strongest_building_name:
      description: >-
        The name of the building containing the locator with the strongest
        alarm signal strength.
      type: string
    strongest_location:
      description: >-
        The name of the location given to the locator with the strongest
        alarm signal strength.
      type: string
    strongest_rssi:
      description: >-
        The signal strength of which the strongest locator heard the alarm
        from the pendant.
      type: integer
    strongest_locator_txid:
      description: >-
        TXID (transmitter ID) of the strongest locator that heard the
        alarm from the pendant.
      type: integer
    alternate_building_name:
      description: >-
        The name of the building containing the locator with the second
        strongest alarm signal strength.  This field will only be filled
        in when a second location has similar signal strength to the
```

```
        strongest location.
      type: string
    alternate_location:
      description: >-
        The name of the location given to the locator with the second
        strongest alarm signal strength.   This field will only be filled
        in when a second location has similar signal strength to the
        strongest location.
      type: string
    alternate_rssi:
      description: >-
        The signal strength of which the second strongest locator heard
        the alarm from the pendant.
      type: integer
    alternate_locator_txid:
      description: >-
        TXID (transmitter ID) of the second strongest locator that heard
        the alarm from the pendant.
      type: integer
    strongest_locator_floor_name:
      description: >-
        The name of the floor of the locator with the strongest
        alarm signal strength (if it exists).
      type: string
    strongest_locator_floor_id:
      description: >-
        The UUID of the floor of the locator with the strongest
        alarm signal strength (if it exists).
      type: string
    strongest_locator_unit_name:
      description: >-
        The name of the unit of the locator with the strongest
        alarm signal strength (if it exists).
      type: string
    strongest_locator_unit_id:
      description: >-
        The UUID of the unit of the locator with the strongest
        alarm signal strength (if it exists).
      type: string
    alternate_locator_floor_name:
      description: >-
        The name of the floor of the locator with the second strongest
        alarm signal strength (if it exists).
      type: string
    alternate_locator_floor_id:
      description: >-
        The UUID of the floor of the locator with the second strongest
        alarm signal strength (if it exists).
      type: string
    alternate_locator_unit_name:
      description: >-
        The name of the unit of the locator with the second strongest
        alarm signal strength (if it exists).
      type: string
    alternate_locator_unit_id:
      description: >-
        The UUID of the unit of the locator with the second strongest
        alarm signal strength (if it exists).
      type: string
device_status:
  summary: >-
    Device status message with the status information for a device on a
```

```
    site.  Currently only used to indicate inactive locators, repeaters, and
    gateways.  Statuses other than Inactive/Active are only available for
    sites in a Senior Living organization.
  payload:
    type: object
    properties:
      status:
        description: The status of the device e.g. 'INACTIVE'
        type: string
        enum:
          - INACTIVE
          - ACTIVE
          - LOW_BATTERY
          - RESET
          - TAMPER
          - LOSS_OF_POWER
          - JAMMED
          - ALARM0
          - ALARM1
          - ALARM2
          - ALARM3
          - EOL_TAMPER
          - TEST_ALARM
          - CLEAN_ME
          - DEFECTIVE_SENSOR
          - COMBINED_FAULT
          - FALL
          - ALL_CLEAR
          - RESTORED
      send_time:
        description: >-
          An ISO8601 string in UTC at which device information was
          published.
        type: string
      last_sync_time:
        description: >-
          An ISO8601 string in UTC at which the device last synced with the
          cloud.
        type: string
      device_txid:
        description: >-
          TXID (transmitter ID) of the locator or repeater that the message
          pertains to (left blank if gateway).
        type: integer
      device_location:
        description: The location of the device that the message pertains to.
        type: string
      device_note:
        description: The note field of the device.
        type: string
      device_model:
        description: The model of the device.
        type: string
      device_type:
        description: The type of device that the message pertains to.
        type: string
        enum:
          - GATEWAY
          - REPEATER
          - LOCATOR
          - TRANSMITTER
      organization_name:
```

```
            description: The name of the organization that the device belongs to.
            type: string
          site_name:
            description: The name of the site that the device belongs to.
            type: string
          building_name:
            description: >-
              The name of the building that the device is assigned to (if
              applicable, left blank otherwise).
            type: string
          rfmessage_id:
            description: >-
              The RFMessage ID of the actual RF Message that triggered the
              alert.
            type: string
  parameters:
    site_id:
      description: >-
        ID of the site. This can be found in the exported configuration file of
        the site.
      schema:
        type: string
    username:
      description: 'Username to authenticate with the MQTT broker, configured for each
site.'
      schema:
        type: string
```