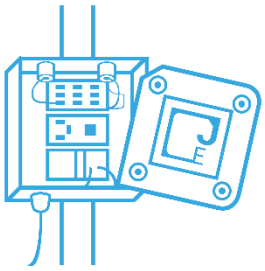
JALUD EMBEDDED

SED REST API Driver

v 4.2



Content

Introduction

The REST API is a standardised format for communication between applications, processes, or between servers and clients. More information about the REST interface itself can be found on websites, e.g. Wikipedia:

The SED server serves as a central hub for communication between SED devices and users. It informs users about device state changes and, most importantly, alerts them to warnings detected by the devices.

The REST API driver is designed as a lightweight version of communication via the ONVIF protocol. Unlike the ONVIF protocol, it cannot send system configuration requests; the driver is narrowly focused on obtaining information about devices and alerts.

User authentication

User authentication when using the REST API driver is done via username and password for each request.

Implemented authentication methods:

- Basic

Planned authentication methods for Q1 2025:

- Digest
- SSL X509 Certifikát

Requests

The server has a defined list of requests it can handle. These requests are used to retrieve information about devices, subscribe to events, or obtain notifications and alerts.

The server's response to a request can be as follows:

- 200 (OK): The request was successfully processed, the requested information is included in the response, or the requested command was executed.
- 401 (Unauthorized): The user was not authorized; authentication credentials are missing or invalid.
- 403 (Forbidden): The user does not have access to the requested data or the data does not exist.
- 404 (Not Found): The user is not subscribed to events, or the subscription no longer exists.

More information about individual requests can be found in the respective subsections or the attached configuration YAML file.

Devices

The server provides users with information about devices assigned to them. Users can obtain information about a specific device by its serial number or about all devices at once.

The SED Server response contains the following device information:

- device_id: Internal unique device ID
- serial: Device serial number.
- location: Device location description defined by the user.
- latitude, longitude: GPS coordinates of the device's location on a map.
- interior: Device location using interior maps.
- x,y: x and y positions on the interior map.
- floor: The floor where the device is located.
- model: Detection model version. The model name always starts with a timestamp in the YYMMDD format. For optimal functionality, the detection model on the device should not be older than six months.

Device list: */devices GET*

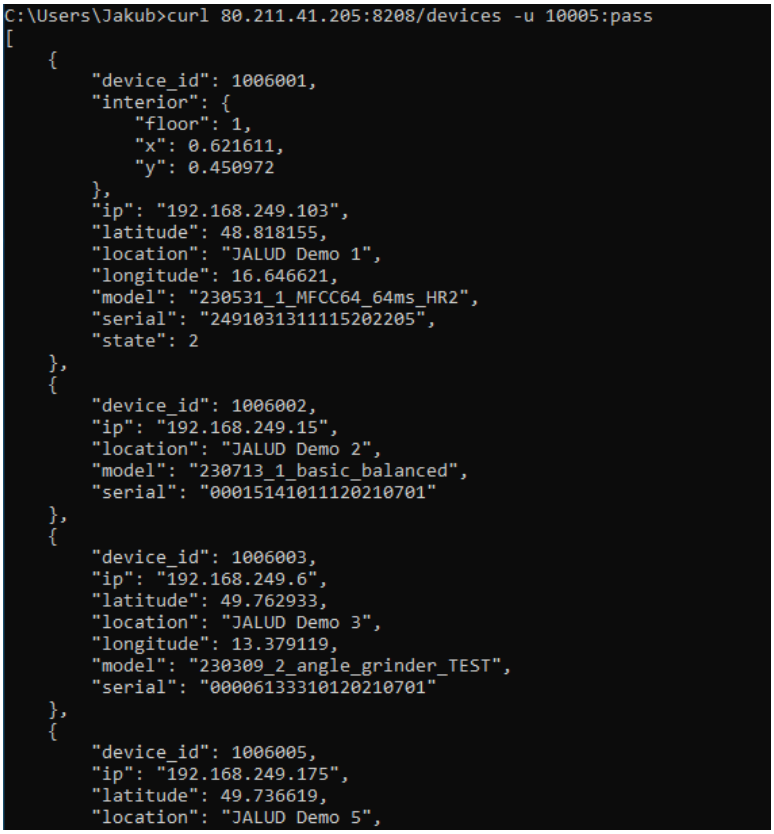


Figure 1: Get devices

Device information: */devices?serial={serial} GET*

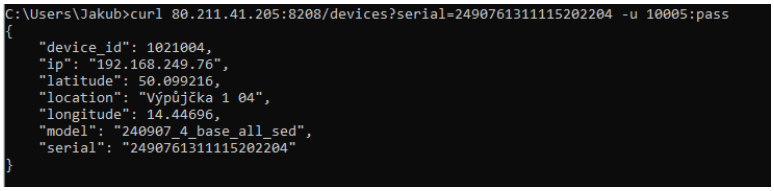


Figure 2: Get device

Device information: */device/{serial} GET*

Figure 3: Get device

Users

The server provides user profile information. The level of detail and search scope depend on the requesting user's role.

The response from the SED Server contains the following user information:

- userId: internal unique ID of the user.
- userName: user's registration name.
- userType: type of user (role), e.g., AdminUser, SuperUser, CommonUser.
- emailAddress: user's email address.
- deviceIds: list of unique device IDs assigned to the user.

User List: */users GET*

This request returns a list of user profiles. Filtering can be applied using the query parameters `id` or `email`.

Access permission mapping:

- AdminUser: Returns all users registered in the system.
- SuperUser: Returns all users who belong to the same group (`mainId`) as the requesting user.
- CommonUser: Returns a list containing only the requesting user's own profile.

Single User Info: */user/{id} GET*

This request retrieves the detailed profile of a single user by their unique ID.

Access permission mapping:

- AdminUser: Returns the requested user profile if it exists; otherwise returns `404 Not Found`.
- SuperUser: Returns the requested user profile if they belong to the same group (`mainId`); otherwise returns `403 Forbidden` (prevents user enumeration).
- CommonUser: Returns their own profile if `{id}` matches their own `userId`; otherwise returns `403 Forbidden`.

Response Payload Example:

```
{
  "userId": 1234,
  "userName": "johndoe",
  "userType": "SuperUser",
  "emailAddress": "john.doe@company.com",
  "deviceIds": [1001001, 1001002]
}
```

Event Subscription

To receive events and alerts, users must first subscribe to events. This method is a lightweight version of the pull-point mechanism in the ONVIF communication protocol.

The user subscribes to events with a request. The server responds with a name that the user will use to renew the subscription or retrieve events. In the current version, users can only subscribe to all events at once.

Detected alerts often include up to a 1.5-second-long audio sample to help the operator verify the alert. If the user does not wish to receive audio recordings for any reason, audio transmission can be disabled by using the **multimedia=none** parameter in the query.

In the next version, there is a plan to save audio recordings in cloud storage and send the user only a download link. The current version does not support this yet.

Subscribe to events: */subscriptions/new POST*

Figure 4: New subscription

The subscription validity is 60 seconds unless renewed by the user or terminated. Each renewal extends the subscription life by 60 seconds.

Renew subscription: */subscriptions/{subscription_name}/renew POST*

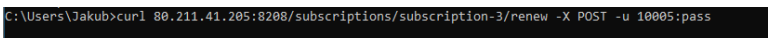


Figure 5: Renew subscription

Unsubscribe from events: *subscriptions/{subscription_name}/unsubscribe POST*

Figure 6: Unsubscribe

Events

After subscribing to events, users can request a list of events. The pull-point mechanism works such that the user sends a request with a defined timeout ranging from 10 to 60 seconds (default is 10 seconds). If an event occurs within the timeout period, the server responds with the event. Otherwise, it waits until the timeout expires and returns a response with an empty event list. After the server response, the user must immediately send a new request

Retrieve events: */subscriptions/{subscription_name}/pull_events?timeout={value} POST*

Figure 7: Retrieve events

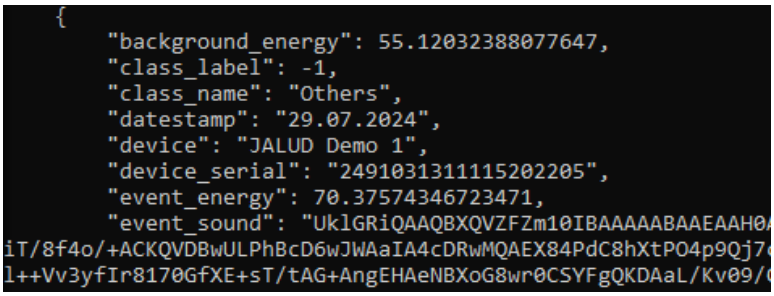


Figure 8: An event

The server returns any event type that occurs during the request period. Events have a common parameter, **EventType**, which specifies the event type. Currently, events are always one of the following types:

- Device State Change
- Detected Alert
- Detected Event
- Alert Localization
- Noise Measurement

Device State Change

This event informs users about changes in device connection status. There are four states:

- Connected: The device is connected.
- Disconnected: The device is disconnected.
- Signal Loss: The signal to the device was temporarily lost. If the device does not respond for several minutes, it will be marked as disconnected. Alerts from this device may be delayed.
- Outdated SW: The software on the device is outdated. Contact us for an upgrade.

Event parameters:

- event_type: "Device State"
- device: Description of the device that changed state
- device_serial: Device serial number
- datestamp, timestamp: Timestamp of the state change
- device_state: New device state

Detected Alert

This event alerts users to a detected dangerous or unwanted event near the device. Alerts can include:

- Gunshot
- Glass break
- Scream, Male scream
- Aggression
- Spray
- Grinder
- Vehicle
- Fireworks, Firecracker
- Siren
- Drone

Event parameters:

- event_type: "Alert"
- device: Description of the device that changed state

- device_serial: Device serial number
- class_name: Name of the detected event type
- class_label: Internal designation of the detected event type
- class_prob: Probability with which the detection model identified the event type
- background_energy: Background noise level
- event_energy: Detected event noise level
- event_sound: Audio recording for alert verification in WAV format. The maximum audio signal length is 1.5 seconds. The recording is encoded in Base64.
- event_pictures: Array of images from the camera module in JPEG format, each encoded in Base64.
- datestamp, timestamp: Timestamp of the state change

Alert Table		
Alert	Class name	Class Label
Gunshot	Gunshot	1
Glassbreak	Glassbreak	6
Glass Tester	Glass Tester	8
Scream	Scream	11
Male scream	Male Scream	12
Aggression	Aggression	13
Fireworks	Fireworks	18
Grinder	Grinder	19
Siren	Siren	20
Spray	Spray	21
Vehicle (engine)	Vehicle	22
Vehicle (tire)	Traffic	23

Drone	Drone	25

Detected Event

In addition to dangerous or unwanted events, devices can also recognize ordinary events that are usually not sent to the user as they are not important. In specific cases, sending these events can be enabled as they may indicate undesirable behavior, such as nighttime disturbances or tire screeching during reckless driving etc.

Events can include:

- People
- Children
- Music
- Animals
- Birds
- Honking
- Screeching: Brake screeching, tire squealing
- Alarm: e.g., car alarm, alarm clock
- Attention: Doorbell, bell, car backup beep
- Construction work
- Transportation
- Others

Event parameters:

- **event_type:** "Event"
- **device:** Description of the device that changed state
- **device_serial:** Device serial number
- **class_name:** Name of the detected event type
- **class_label:** Internal designation of the detected event type
- **class_prob:** Probability with which the detection model identified the event type
- background_energy: Background noise level
- event_energy: Detected event noise level

- event_sound: Audio recording for event verification in WAV format. The maximum audio signal length is 1.5 seconds. The recording is encoded in Base64.
- datestamp, timestamp: Timestamp of the state change

Event Table		
Alert	Class name	Class Label
Others	Others	-1
Bird	Bird	-2
People	People	-3
Children	Children	-4
Music	Music	-5
Construction	Construction	-7
Creak	Creak	-8
Attention	Attention	-9
Animal	Animal	-10
Transport	Transport	-14
Horn	Horn	-15

Event Localization

If the same event is detected by at least three devices simultaneously, the approximate position of the event source can be calculated from the time stamps. For example, the shooter's position in the case of gunfire, or the victim's position in the case of screaming

Event parameters:

- TBD

Noise Measurement Event

This event informs users about the ambient noise level around the device.

Event parameters:

- event_type: "Measurement"
- device: Description of the device that changed state
- device_serial: Device serial number
- background_energy: Ambient noise level
- event._energy: Detected event noise level
- datestamp, timestamp: Timestamp of the state change