MYDEFENCE

Counter Drone Technolo

ARGOS

Interface Specification

MYDEFENCE

v6.0.0 2025.05

About MyDefence

MyDefence was founded by military officers with insight into military operations and advanced radio technology. We specialize in developing, producing and selling anti-drone products for detection, tracking and jamming of Unmanned Aerial Systems (UAS) to mitigate the threat of malicious drones.

MyDefence protects those who protect us. Our field proven products provide end-users with state-of-the-art technology for enhanced protection and situational awareness. By listening to our end-users and combining their learnings with our technology, we are producing innovative and versatile Counter UAS solutions for any type of scenario.

Table of Contents

1. Introduction

This document serves as a specification to the developer who wants to communicate with the ARGOS API, to control networked devices or get information or data related devices or mission setup.

The version of the specification follows the version of the ARGOS backend, and information about changes is found in the accompanying release note.

The newest version of this document along with additional source code examples and application notes can be found on our github page at `https://github.com/mydefence/argos_api`.

2. Concept

ARGOS main purpose is to simplify the communication between client applications (Backend/Frontend) and the attached physical devices. Furthermore, it is also the purpose of ARGOS to store and enrich data collected or calculated by one of the services inside of ARGOS or from a connected 3rd part.

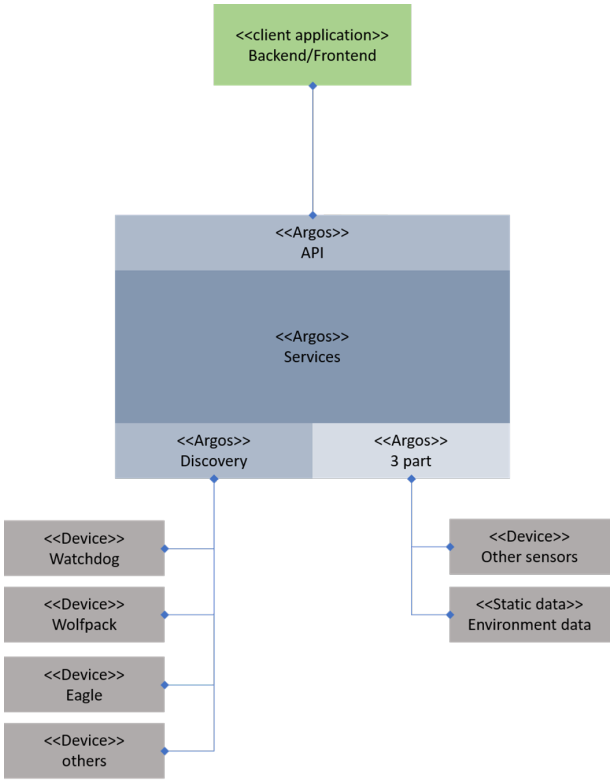

Figure 2.1: Conceptual model of ARGOS system.

ARGOS is internally built as a micro-service architecture so that it can be customized with different features and in different scales, depending upon the individual use case.

ARGOS consists of 4 high level elements which takes care of the following things:

- API exposes the event handles available for communicating with the attached devices and the stored and enriched data.
- Services is a bundle of micro-services that takes care of communicating, collecting, storing, analyzing, enriching data and communication.
- Discovery make sure that any recognized device is automatically attached to ARGOS and communicate if any device is removed or connection is lost.
- 3rd party handles communicate with 3rd party applications and the ARGOS micro-services.

2.1 Connecting to ARGOS API

The standard ARGOS API use a real time, bi-directional bus for communication named socket.IO. Socket.IO primarily uses the WebSocket protocol with polling as a fallback option.

ARGOS API can be reached on port **5050** (HTTP) and port **5051** (HTTPS). The HTTP port is default disabled for security reasons, but can be enabled in the system manager on port **8080**. The IP address is that of the host machine. If the IP address is not known, it can be looked up using the mDNS name *c2-server.local*. To use HTTPS you must install an additional root CA on the client PC. This can be downloaded from the system manager.

Normally user authentication is required for access to Argos. See chapter 4 on page 20 for more information. User authentication can also be disabled in the system manager.

The typical use-case for the ARGOS API is illustrated in the message sequence diagram in fig. 2.2 on page 11.

2.2 JSON Schema

All ARGOS messages are defined according to the `http://json-schema.org/draft-07/schema` standard. For each of the messages described in this document a schema and sample of the document can be found in the available ARGOS-schema archive.

In addition, the schemas are used to generate typescript types, which can be located in the documentation package (interface_examples directory).

2.3 Message structure

All messages are wrapped into a common message structure when send and received from the ARGOS API interface. Each individual event will use this message structure and handle the specific details for the specific event like event (name) and message payload. For more information about the individual events can be found in the following sections.

ARGOS employs two message concepts - "request/response" and "push". Push messages are sent from ARGOS to all connected clients when an event occurs, e.g. a threat is started («threatStarted»). They are sent as an event named as the message name. A client should set up permanent listeners for each push message it is interested in. Push message events can occur at any time. Request/response messages are initiated by the client. When sending a request to ARGOS, the client must set the "responseId" field in the API message object. ARGOS will reply with a response to the

client sent as an event named as this "responseId". After sending the request the client should setup a one-time listener for the event named by the "responseId". In the message sequence charts in this interface specification push messages are set in regular font and request/response messages are set in italic font. Figure 2.2 on the following page shows this, where all request/response messages flows between ARGOS and Client 1, and push messages flows to both connected clients.

For schema and samples, see file in ARGOS-schema archive: `schema/Socket_io_message.json`.

Messages not complying to the schema are rejected with an «apiError» message: `schema/messages/ApiError.json`

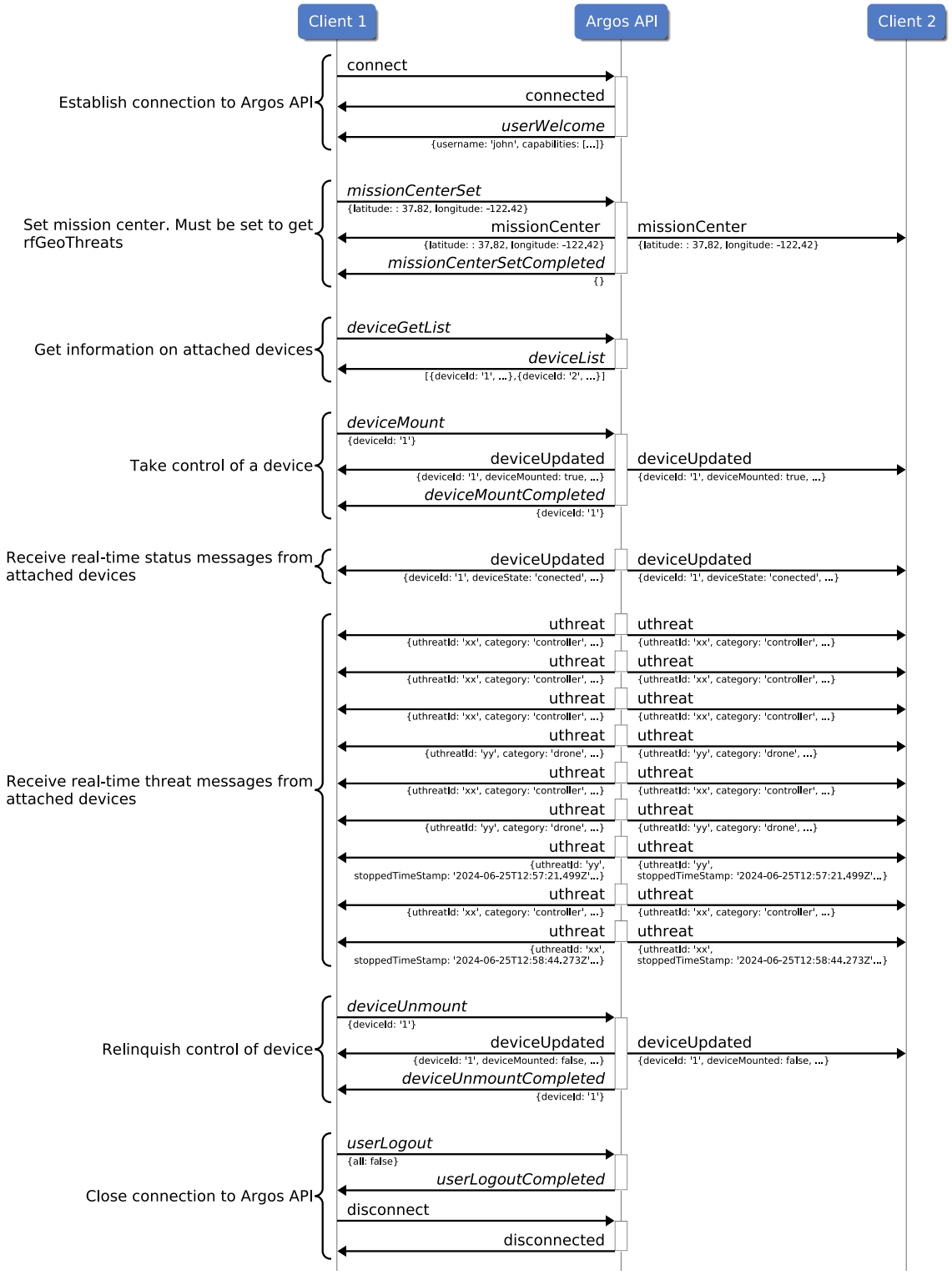

Figure 2.2: Example of connection and message exchange between client and ARGOS API.

3. Event overview

The ARGOS API supports several events to discover and control the device(s), detect threats and communicating with ECM devices. Each type of event is implemented either using a push pattern or a request/response pattern.

3.1 Authentication

These events are related to logging in and out of the system.

Request	Push / Response	Type	Description
	userWelcome	Response(*)	Sent after successful authentication
userLogout	userLogoutComplete	Request / Response	Log out current user

3.2 Discovering a device

These events are common for all devices.

Request	Push/Response	Type	Description
	deviceAdded	Push	Information about the device is added
	deviceRemoved	Push	Information about the device is removed
	deviceUpdated	Push	Information about the device is updated
	deviceList	Push	A list of all currently attached devices
deviceGetList	deviceList	Request/Response	A list of all currently attached devices
deviceAdd	deviceAddCompleted deviceAddError	Request/Response	Used when adding a device manually
deviceRemove	deviceRemoveCompleted deviceRemoveError	Request/Response	Used when a device needed to be removed

3.3 Controlling a device

These events are common for all devices.

Request	Push/Response	Type	Description
deviceMount	deviceMountCompleted deviceMountError	Request/Response	Taking control of the device deviceMountError and start receiving threat information when threats are detected
deviceUnmount	deviceUnmountCompleted deviceUnmountError	Request/Response	Relinquish control of the device and stop receiving threat information
deviceLocationChange	deviceLocationChangeCompleted deviceLocationChangeError	Request/Response	Placing a device in a geolocation and setting the direction of the device
deviceMiscellaneousChange	deviceMiscellaneousChangeCompleted deviceMiscellaneousChangeError	Request/Response	Modifying one of the miscellaneous attributes related to a device
deviceGetLocations	deviceLocations	Request/Response	A list of all device locations for a specific device
deviceGetMiscellaneous	deviceMiscellaneous	Request/Response	A list of all device miscellaneous attributes for a specific device
	deviceLocationChanged	Push	Information about the device location is updated
	deviceMiscellaneousChanged	Push	Information about the device miscellaneous data is updated

3.4 Receiving uthreats

These events are specifically for uthreats.

Request	Push/Response	Type	Description
	uthreat	Push	Information about a threat
	uthreatConfiguration	Push	Configuration of the uthreat interface
	uthreatList	Push	All active threats
uthreatConfigure	uthreatConfigureCompleted uthreatConfigureError	Request/Response	Configure the uthreat interface
uthreatGetConfiguration	uthreatConfiguration	Request/Response	Get configuration
uthreatGetList	uthreatList	Request/Response	Get all active threats
uthreatSetMute	uthreatSetMuteCompleted uthreatSetMuteError	Request/Response	Mute/unmute a specific uthreat
uthreatGetHistory	uthreatHistory uthreatGetHistoryError	Request/Response	An overview of uthreats for a given time frame
uthreatGetHistoryData	uthreatHistoryData uthreatGetHistoryDataError	Request/Response	A detailed list of updates for a given uthreat

3.5 Receiving alarms

These events are specifically for alarms caused by geo-located threats that enter or leave alarm zones.

Request	Push/Response	Type	Description
	alarmStarted	Push	Information about a new alarm is detected
	alarmStopped	Push	Information about an existing alarm is stopped
	alarmUpdated	Push	Updated information about an existing alarm

3.6 Communicating with ECM

These events are specifically for ECM (Electronic Countermeasures) devices.

Request	Push/Response	Type	Description
ecmStart	ecmStartCompleted ecmStartError	Request/Response	Starting an ECM device/asset
ecmStop	ecmStopCompleted ecmStopError	Request/Response	Stopping an ECM device/asset
ecmGetState	ecmState	Request/Response	Getting the current state of an ECM device/asset
ecmConfigure	ecmConfigureCompleted ecmConfigureError	Request/Response	Configure ECM settings
ecmGetHistory	ecmHistory	Request/Response	An overview of ecmStart events for a given time frame
	ecmUpdated	Push	Current state of an ECM device/asset
	ecmConfiguration	Push	Current ECM configuration

3.7 RF threat types

These events are specifically handling RF threat type information.

Request	Push/Response	Type	Description
	threatTypeList	Push	Current list of threat types
threatTypeGetList	threatTypeList	Request/Response	Get list of threat types
threatTypeMute	threatTypeMuteCompleted threatTypeMuteError	Request/Response	Mute a specific threat type
threatTypeUnMute	threatTypeUnMuteCompleted threatTypeUnMuteError	Request/Response	Unmute a specific threat type

3.8 Alarm zones

These events are specifically for setting up, manipulating and removing alarm zones

Request	Push/Response	Type	Description
	alarmZoneAdded	Push	An alarm zone has been created
	alarmZoneUpdated	Push	An alarm zone has been updated
	alarmZoneRemoved	Push	An alarm zone has been Removed
	alarmZoneList	Push	List of alarm zones
alarmZoneAdd	alarmZoneAddCompleted alarmZoneAddError	Request/Response	Add an alarm zone
alarmZoneUpdate	alarmZoneUpdateCompleted alarmZoneUpdateError	Request/Response	Update an alarm zone
alarmZoneRemove	alarmZoneRemoveCompleted alarmZoneRemoveError	Request/Response	Remove an alarm zone
alarmZoneGetList	alarmZoneList	Request/Response	Get list of alarm zones

3.9 PTZ

These events are specifically for controlling PTZ (pan-tilt-zoom) devices.

Request	Push/Response	Type	Description
ptzGetDeviceList	ptzDeviceList	Request/Response	Get list of PTZ devices
	ptzDeviceInfo	Push	PTZ device information
	ptzDeviceUpdate	Push	Change to PTZ device state
ptzSetVelocity	ptzSetVelocityCompleted	Request/Response	Turn PTZ device in a specific direction
ptzMoveAbs	ptzMoveAbsCompleted	Request/Response	Turn PTZ device in a specific direction
ptzDeviceSet	ptzDeviceSetCompleted	Request/Response	Change PTZ setting
	ptzFollowStarted	Push	PTZ follow instance has been started
	ptzFollowStopped	Push	PTZ follow instance has been stopped
	ptzFollowList	Push	List of PTZ follow instances
ptzFollowStart	ptzFollowStartCompleted ptzFollowStartError	Request/Response	Follow a device
ptzFollowStop	ptzFollowStopCompleted ptzFollowStopError	Request/Response	Stop following a device
ptzFollowGetList	ptzFollowList	Request/Response	Get list of PTZ follow instances

3.10 Firmware update

These events are specifically for adding FWU images and performing FWU.

Request	Push/Response	Type	Description
	fwuImageAdded	Push	An FWU image has been added
	fwuImageRemoved	Push	An FWU image has been removed
	fwuImageList	Push	List of FWU images
	fwuUpdateableList	Push	List of updateable devices
fwuImageAdd	fwuImageAddCompleted fwuImageAddError	Request/Response	Add an FWU image
fwuImageRemove	fwuImageRemoveCompleted fwuImageRemoveError	Request/Response	Remove an FWU image
fwuImageGetList	fwuImageList	Request/Response	Get list of FWU images
fwuUpdateableGetList	fwuUpdateableList	Request/Response	Get list of updateable devices
fwuStart	fwuStartCompleted fwuStartError	Request/Response	Start FWU process for device(s)

3.11 System state

These events are specifically used for informing the API client of the system state.

Request	Push/Response	Type	Description
	systemState	Push	Information about system state
getSystemState	systemState	Request/Response	Information about the system state
	time	Push	Information about the server time
getTime	time	Request/Response	Information about the server time

3.12 rfGeoThreat simulation

These events are specifically used for simulating RF geo-location coverage for a given setup.

Request	Push/Response	Type	Description
rfGeoThreatSimulationStart	rfGeolocationSimulationStartCompleted rfGeolocationSimulationStartError	Request/Response	Start coverage simulation
	rfGeoThreatSimulationResult	Push	Coverage simulation results

3.13 Mission Center Handling

These events are used to get and set mission center.

Request	Push/Response	Type	Description
	missionCenter	Push	Current mission center
missionCenterSet	missionCenterSetCompleted	Request/Response	Set mission center
missionCenterGet	missionCenter	Request/Response	Get mission center

3.14 Video Management System

These events are used to handle video.

Request	Push/Response	Type	Description
	vmsStreamList	Push	List of current video streams
	vmsBoundingBox	Push	Threat bounding box information
vmsStreamGetList	vmsStreamList	Request/Response	Get list of video streams
vmsSdpInit	vmsSdpInitCompleted	Request/Response	WebRTC initialization

3.15 Features

These events are specifically used for informing the API client about the available features.

Request	Push/Response	Type	Description
	featureList	Push	List of features
featureGetList	featureList	Request/Response	Get list of features

3.16 Origin Filters

These events are specifically for setting up, manipulating, and removing radar origin filters.

Request	Push / Response	Type	Description
originFilterAdd	originFilterAddCompleted originFilterAddError	Request / Response	Add an origin filter
originFilterUpdate	originFilterUpdateCompleted originFilterUpdateError	Request / Response	Modify an existing origin filter
originFilterRemove	originFilterRemoveCompleted originFilterRemoveError	Request / Response	Remove an origin filter
	originFilterAdded	Push	The origin filter has been added
	originFilterUpdated	Push	The origin filter has been modified
	originFilterRemoved	Push	The origin filter has been removed
originFilterGetList	originFilterList	Request / Response	Get list of origin filters
	originFilterList	Push	List of origin filters

3.17 Sapient

These events are specifically for setting up and monitoring Sapient connections.

Request	Push / Response	Type	Description
sapientConfigure	sapientConfigureCompleted sapientConfigureError	Request / Response	Configure Sapient connections
sapientGetConfiguration	sapientConfiguration	Request / Response	Get Sapient connection configuration
	sapientConfiguration	Push	Sapient connections has been configured
sapientNodeGetList	sapientNodeList	Request / Response	Get list of Sapient nodes
	sapientNodeList	Push	List of Sapient nodes
	sapientNodeUpdated	Push	Sapient node has been updated

3.18 TAK

These events are specifically for setting up and monitoring connections to TAK servers.

Request	Push / Response	Type	Description
takConfigure	takConfigureCompleted takConfigureError	Request / Response	Configure TAK server connections
takGetConfiguration	takConfiguration	Request / Response	Get TAK server connection configuration
	takConfiguration	Push	TAK server connections has been configured
takTransportGetList	takTansportList	Request / Response	Get list of TAK transports
	takTransportList	Push	List of TAK transports
	takTransportUpdated	Push	TAK transport has been updated

3.19 System Log

These messages are notifications of system events.

Request	Push / Response	Type	Description
	systemLog	Push	A System Log event has been pushed
systemLogGetList	systemLogList	Request / Response	Get list of active system log events
	systemLogList	Push	A System Log event is pushed upon service start

4. Authentication

ARGOS uses a token-based authentication according to best practices for web based authentication.

4.1 Login Procedure

Login is a 2 step process:

1. The user sends a login request to the login service accessible on the same URI as the Argos service, e.g. HTTP POST *https://c2-server.local:5051/login* with JSON encoded payload containing `username` and `password`. The system checks the username and password, and if they are correct, generates a token and sends it back to the user in a httpOnly cookie.
2. The user/browser sends the token cookie in the header of the HTTP request to the ARGOS API when creating the socket.io connection. Argos checks the token and if it is valid, the user is authenticated and the «userWelcome» event is sent.

The token is valid for a limited time (configurable in the system manager). If the token is expired, the user must login again. In a web application the token is stored in a cookie, so login is not needed on each access to Argos.

4.2 userWelcome

After a successful login, the ARGOS API sends the «userWelcome» event to the client. This event contains the username and the capabilities of the user. See Section 4.6 for more details.

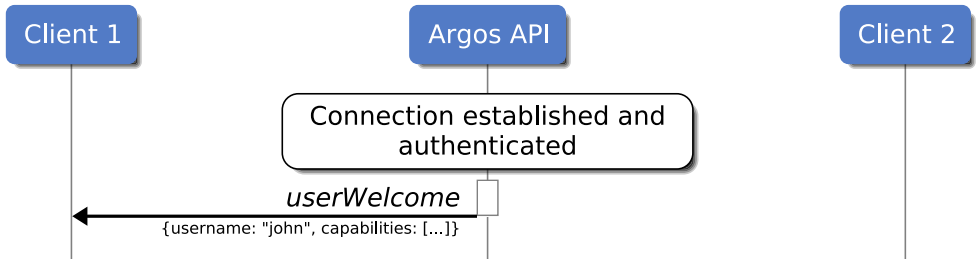

Figure 4.1: Example of connection and message exchange between client and ARGOS API.

Response: userWelcome
«userWelcome» is sent after successful authentication.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/UserWelcome.json`

Note, that this response does not follow the standard response structure as described in section 2.3, as it is not sent to a "responseId", but is sent like the push messages, but only on this single client.

4.3 userLogout

Pattern: Request/Response

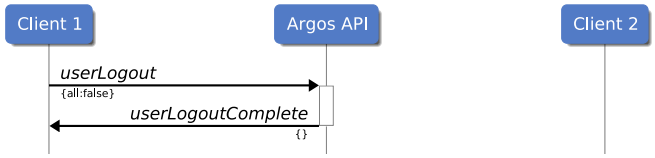

Figure 4.2: Example of userLogout message communication between the client and the ARGOS API.

Use «userLogout» to log out the current user and invalidate the token. If you want to keep the token for a future session, close the connection without logging out.

When using the «userLogout» command a response with «userLogoutComplete» will be sent back as response.

Request: userLogout
«userLogout» may be sent at any time. All following commands will be rejected.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/UserLogout.json`

Response: userLogoutComplete
When «userLogout» request command is successfully registered it will respond with the «userLogoutComplete» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json`

4.4 User Management

User management can only be performed in the system manager. The system manager is a web application running on the same host typically port 8080, so it is available on *https://c2-server.local:8080/*. The system allows administrators to create, delete, and modify user accounts, as well as assign roles and permissions.

4.4.1 Default Users

The system has the following predefined users. They may be deleted or modified. It is highly recommended to change the passwords of the predefined users.

Username	Role	Password	Description
admin	Administrator	SysAdmin	Has full access to all system features and settings. Admin user cannot be removed, but password may be changed
operator	Default operator	(none)	Can access and use basic features of the front-end but cannot configure the system.
setup	Default super user	iris	Has full access to Argos including configuration, but not to system manager.
argos	Fallback if no authentication	(none)	Default disabled. Will be used to login if no token cookie is supplied. This only works if the user has been enabled and has no password.

4.5 Disabling Authentication

Authentication can be disabled in the system manager. This is not recommended, as it will allow anyone to access the system without any restrictions. For testing or backwards compatibility purposes, authentication can be disabled by enabling the special user *argos* without a password. Then the login step may be skipped and when no token cookie is supplied, Argos will automatically attempt to login with the *argos* user.

4.6 User Capabilities

User capabilities are used to define what actions a user can perform in the system. All capabilities are defined in the ARGOS-schema archive:
`schema/messages/common/Capabilities.json`

4.7 C2 User Capabilities

Only the capabilities starting with `CAP_C2_` are relevant in this context.

Capability	Description
CAP_C2_ECM	Allows the user to configure, start and stop ECM (Electronic Counter Measures) operations, except enabling GNSS frequencies.
CAP_C2_ECM_STOP	Allows the user to stop ECM operations. Useful when automatic jamming has been configured.
CAP_C2_ECM_GNSS	Required for requests configuring GNSS jamming, in addition to the listed capability.
CAP_C2_CONFIG	Allows the user to configure other Argos settings.
CAP_C2_MUTE	Allows the user to mute alarms and notifications in the C2 system.

4.8 Requests with Capability Requirements

The following table lists all requests, that require some capability.

Request	**Capability**
deviceAdd	CAP_C2_CONFIG
deviceRemove	CAP_C2_CONFIG
deviceMount	CAP_C2_CONFIG
deviceUnmount	CAP_C2_CONFIG
deviceLocationChange	CAP_C2_CONFIG
deviceMiscellaneousChange	CAP_C2_CONFIG (*)
ecmStart	CAP_C2_ECM
ecmStop	CAP_C2_ECM / CAP_C2_ECM_STOP
ecmConfigure	CAP_C2_ECM (*)
compositeDeviceAdd	CAP_C2_CONFIG
fwuImageAdd	CAP_C2_CONFIG
fwuImageRemove	CAP_C2_CONFIG
fwuStart	CAP_C2_CONFIG
alarmZoneAdd	CAP_C2_CONFIG
alarmZoneRemove	CAP_C2_CONFIG
alarmZoneUpdate	CAP_C2_CONFIG
rfGeoThreatSimulationStart	CAP_C2_CONFIG
threatTypeMute	CAP_C2_MUTE
threatTypeUnMute	CAP_C2_MUTE
missionCenterSet	CAP_C2_CONFIG
uthreatConfigure	CAP_C2_CONFIG
uthreatSetMute	CAP_C2_MUTE
originFilterAdd	CAP_C2_CONFIG
originFilterUpdate	CAP_C2_CONFIG
originFilterRemove	CAP_C2_CONFIG
sapientConfigure	CAP_C2_CONFIG
rfRecordThreatsConfigure	CAP_C2_CONFIG
takConfigure	CAP_C2_CONFIG

(*) if «deviceMiscellaneousChange» configures ECM frequencies the requried capability is CAP_C2_ECM. if «deviceMiscellaneousChange» configures ECM frequencies to include GNSS, or «ecmConfigure» enables auto jamming with GNSS, the user must *also* have capability CAP_C2_ECM_GNSS.

4.9 Source Code Examples

Authentication requires relatively simple code to implement. We provide examples showing how to implement the login and authentication in a javascript web client and a typescript node.js client. Other languages should be similar.

4.9.1 Javascript Web Client

The following will login the user, and store the credentials in a cookie:

```
    await fetch(`https://${serverUrl}/login`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        credentials: 'include',
        body: JSON.stringify({ username, password }),
    })
```

To establish a secure connection using socket.io, you need to set the flags *withCredentials* and *secure*. This will use the cookie stored in the previous step to authenticate:

```
    import { io } from "socket.io-client"
    const socketClient = io(serverUrl, {
    ...options,
    withCredentials: true,
    secure: true,
})
```

If it fails with a *connect_error*, you need to login again.

4.9.2 node.js Client

The example code included with this documentation handles the login in the file *argosClient.ts*. It is also available on our GitHub page at `https://github.com/mydefence/argos_api`.

5. Discovering a device

When ARGOS is running an automatic discovery service will detect if a MyDefence device is attached to the network or if an attached device is detached from the network. This discovery service is by default running and broadcasting UDP messages to discover MyDefence devices. If for some reason this is not an option, it is possible to contact MyDefence to have this service disabled.

Auto discovered composite devices, are added to the device list, when the first child is discovered. I.e. composite devices usually starts out as partially discovered for a short time. They can still be used normally. If you want to know if the composite device is fully discovered, then test that all children have non-null IP address.

5.1 deviceAdded

Pattern: Push

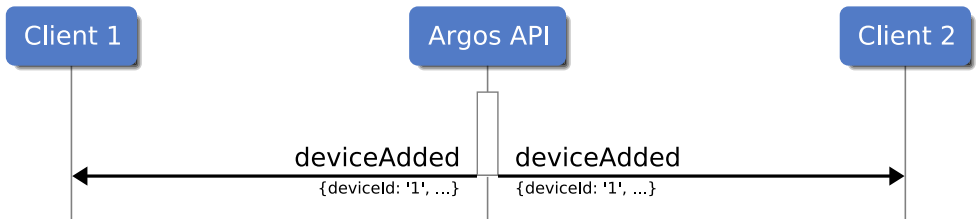

Figure 5.1: Example of deviceAdded message communication from ARGOS API to a connected client when a new device is attached to the network.

When a new MyDefence device is attached to the network a «deviceAdded» message is pushed to the client. The detection cycle is up to 30 second from attaching a MyDefence device to the network, to a «deviceAdded» message is sent to the client. A MyDefence device or supported 3rd-party device can also be attached manually through use of the «deviceAdd» message, which will also result in an «deviceAdded» message.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Device.json`

5.2 deviceRemoved

Pattern: Push

When an added device is removed by e.g. a «deviceRemove» a «deviceRemoved» message is pushed to all connected clients.

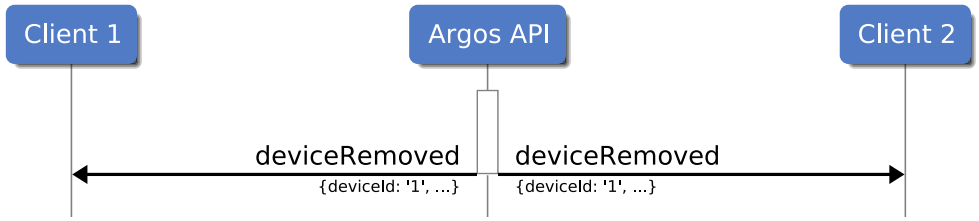

Figure 5.2: Example of deviceRemoved message communication from ARGOS API to a connected client.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Device.json`

5.3 deviceUpdated

Pattern: Push

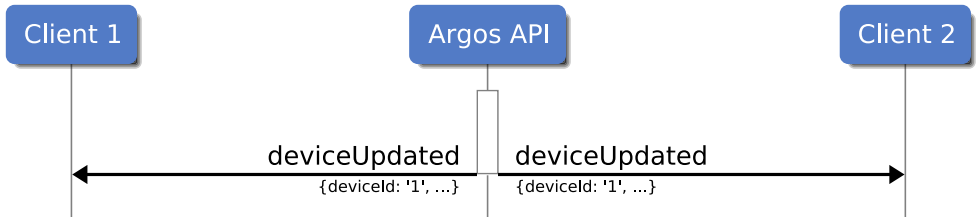

Figure 5.3: Example of deviceUpdated message communication from ARGOS API to a connected client.

When an added device is updated by e.g. a «deviceMount», «deviceUnmount» or another attribute changes a «deviceUpdated» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Device.json`

5.4 deviceGetList

Pattern: Request/Response

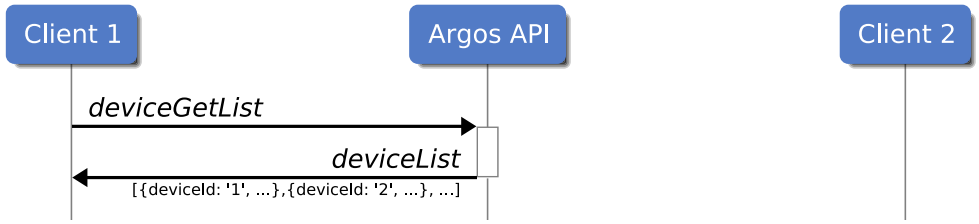

Figure 5.4: Example of deviceGetList message communication between the client and the ARGOS API.

If the full list of all devices added to ARGOS is needed the «deviceGetList» command

must be sent and a response with «deviceList» will be send back with the current list of all devices. Devices that are included in a composite device will not be returned.

Request: deviceGetList
It is possible to send a request to get all devices added to ARGOS.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: deviceList
When a «deviceGetList» request command is successfully registered it will respond with a «deviceList» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceList.json`

6. Controlling a device

When ARGOS has discovered a MyDefence device, it is possible to take control of the device and thereby starting or stopping detection of threats. It is also possible to add the actual geo-location manually if the device does not provide it automatically and virtually rotating the device if not placed directly phasing north. It is also possible to add supported devices manually and creating composite devices from existing devices.

6.1 deviceAdd

Pattern: Request/Response

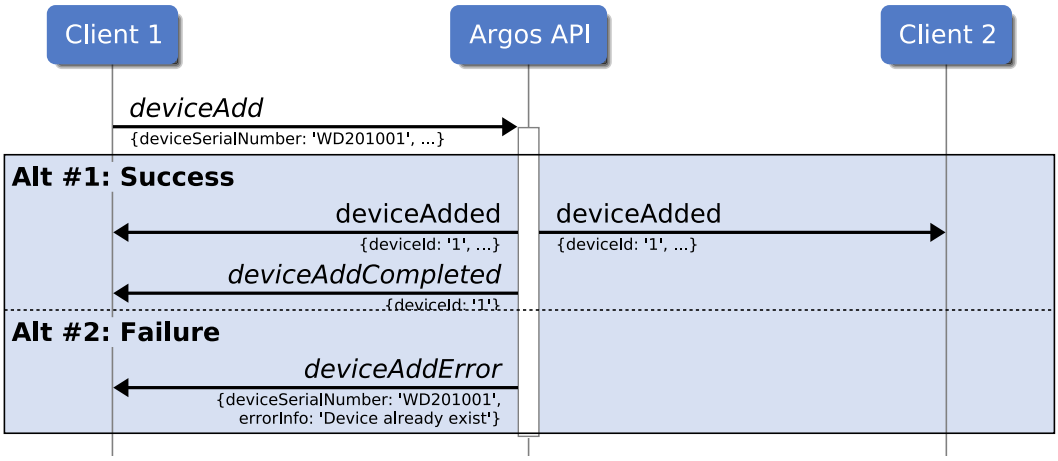

Figure 6.1: Example of deviceAdd message communication between the client and the ARGOS API.

When a supported device is manually added a «deviceAdd» command must be sent and a response with «deviceAddCompleted» will be send back to show that the device has been added. The operation causes a push message with a «deviceAdded» to be sent with information about the device that has been added. In case an error occurs during the «deviceAdd» a «deviceAddError» message will be sent as response.

Request: deviceAdd
It is possible to send a request to create a new supported device and have it added to the list of available devices.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceAdd.json`

When the «deviceAdd» request has been processed a «deviceAdded» message will be pushed with information about the device.

Response: deviceAddCompleted

When a «deviceAdd» request command is successfully registered it will respond with a «deviceAddCompleted» response and if an error occurs a «deviceAddError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/DeviceSerialNumberError.json`

6.2 deviceRemove

Pattern: Request/Response

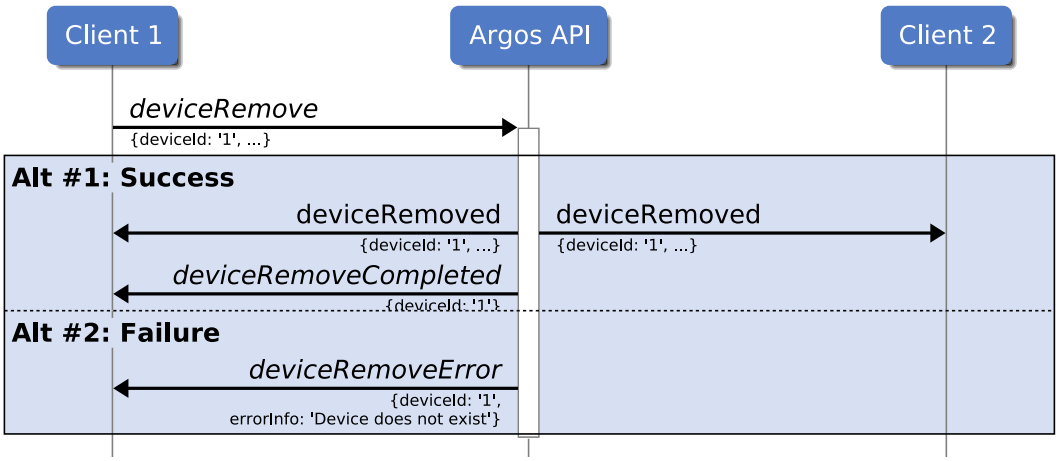

Figure 6.2: Example of deviceRemove message communication between the client and the ARGOS API.

When a device needs to be removed a «deviceRemoved» command must be sent and a response with «deviceRemoveCompleted» will be send back to show that the device has been removed. The operation causes a push message with a «deviceRemoved» to be sent with information about the device that has been removed. In case an error occurs during the «deviceRemove» a «deviceRemoveError» message will be sent as response.

Request: deviceRemove
It is possible to send a request to remove a previous added device. When a «deviceRemove» request is processed all data related to the device is deleted and can not be recovered afterwards. That is both the device itself and any location or miscellaneous data attached to the device.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

When the «deviceRemove» request has been processed a «deviceRemoved» message will be pushed with information about the device.

Response: deviceRemoveCompleted
When a «deviceRemove» request command is successfully registered it will respond

with a «deviceRemoveCompleted» response and if an error occurs a «deviceRemoveError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/DeviceDeviceIdError.json`

6.3 compositeDeviceAdd

Composite devices are used to group multiple co-located physical devices, e.g. on a tower or a vehicle, into one virtual (composite) device. For MyDefence RF sensors, this is required to filter out detections coming from outside the sensor's field of view. (Note, that a WolfPack unit will automatically create a composite device).

Other benefits include:

- Easy to present a single device in your UI.
- Move or rotate all devices only requires moving the composite device.
  - Note, that child devices headings are relative to the composite device.
- Automatic update of position or heading from one of the children will be applied to the entire composite. (Requires useGPSLocation to be set to enabled in deviceMiscellaneous interface.)

Pattern: Request/Response

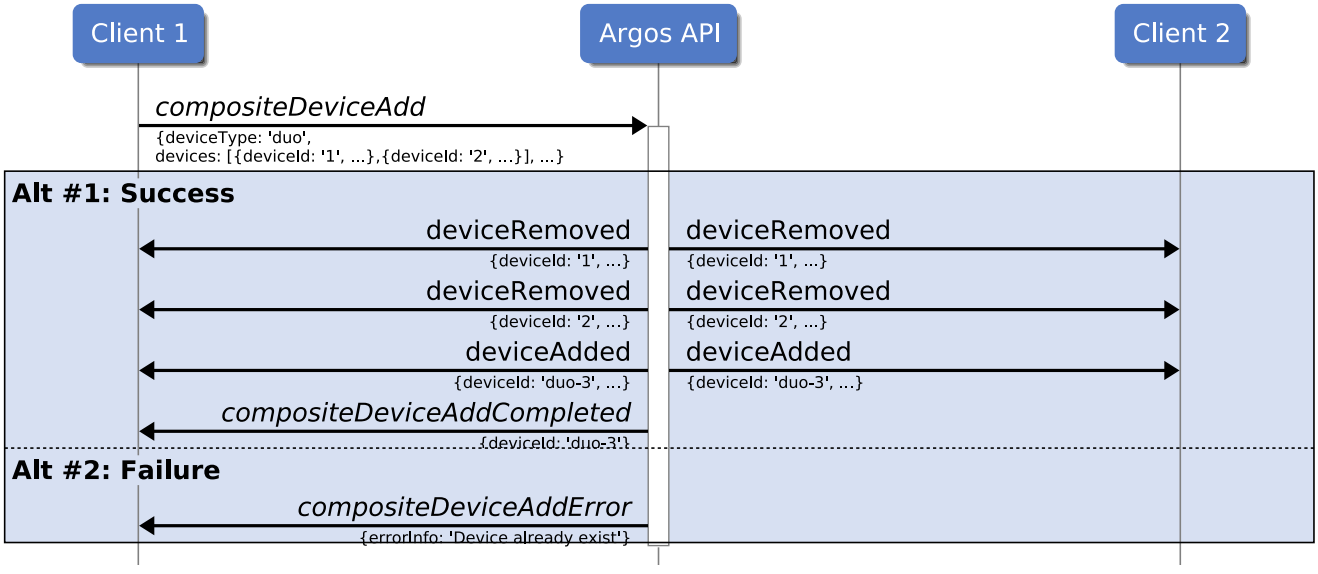

Figure 6.3: Example of message communication between the client and the ARGOS API when adding a new composite device.

When a new composite device is added a «compositeDeviceAdd» request message is sent from the client to the ARGOS API. ARGOS responds with a «compositeDeviceAddCompleted» message. During the add operation a «deviceRemoved» message for each of the devices that should be joined is pushed to all the clients listening and after-

wards a «deviceAdded» message is pushed to all the clients listening to inform about the device just added. In case of error during the operation a «compositeDeviceAddError» message is sent as a response.

Request: compositeDeviceAdd
The request creates a new composite device and the device is added to the list of available devices.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/CompositeDeviceAdd.json`

The deviceType can be set to one of the predefined composite types listed in the schema, in which case you get checks, that the correct device types are used. Alternatively deviceType can be set to the generic "composite" type. If choosing "composite" as type, you may compose any devices you like. Location and heading updates for type "composite" will be taken from the first device in the list with GPS capability. (For a predefined type, GPS source is selected according to the predefined template.) Generic composites may even include existing composites. It is possible to include a partially discovered composite device, but its non-discovered children will be ignored, even if they are later discovered.

When the «compositeDeviceAdd» request has been processed a «deviceAdded» message is pushed to clients containing information about the created device.

Response: compositeDeviceAddCompleted
When a «compositeDeviceAdd» request is successfully handled by the ARGOS API a «compositeDeviceAddCompleted» response is returned to the client. In case of error during the add operation a «compositeDeviceAddError» is returned.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/Error.json`

6.4 deviceMount

Pattern: Request/Response

When a device is discovered it must be mounted before it can start detecting threats. When using the «deviceMount» command a response with «deviceMountCompleted» will be sent back to show that the device has been mounted. The operation causes a push message with a «deviceUpdated» to be sent with information about the device that has been mounted. In case an error occurs during the «deviceMount» a «deviceMountError» message will be sent as response.

Composite devices start with all their children mounted. If a child has been unmounted, it can be mounted again with this command. See 6.5 for more information.

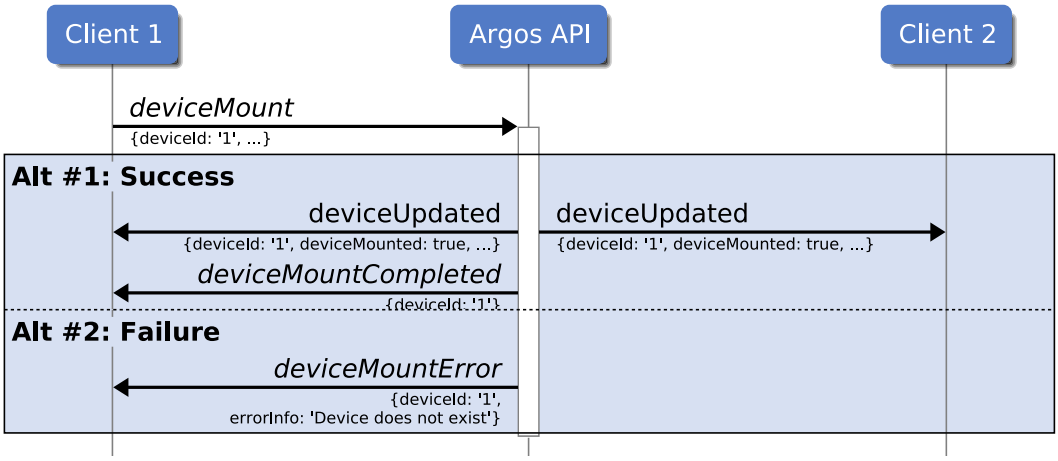

Figure 6.4: Example of deviceMount message communication between the client and the ARGOS API.

Request: deviceMount
The «deviceMount» request command will set the device in a state where it tries to create a connection to the physical device and will continue trying to create the connection until a connection is created or the device is unmounted. If a connection is lost e.g. due to a cable is unplugged, it will try to reconnect to the device again when the device is in a mounted state.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

When the «deviceMount» request has set the state and the process of trying to establish the connection to the physical device a «deviceUpdated» message will be pushed with information about e.g. the deviceMounted state and the deviceState.

Response: deviceMountCompleted
When a «deviceMount» request command is successfully registered it will respond with a «deviceMountCompleted» response and if an error occurs a «deviceMountError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/DeviceDeviceIdError.json`

6.5 deviceUnmount

Pattern: Request/Response

When a device is unmounted is releases the any connection it previously has created to the physical device. Using the «deviceUnmount» command a response with «deviceUnmountCompleted» will be sent back to show that the device has been unmounted. The operation causes a push message with a «deviceUpdated» to be sent with infor-

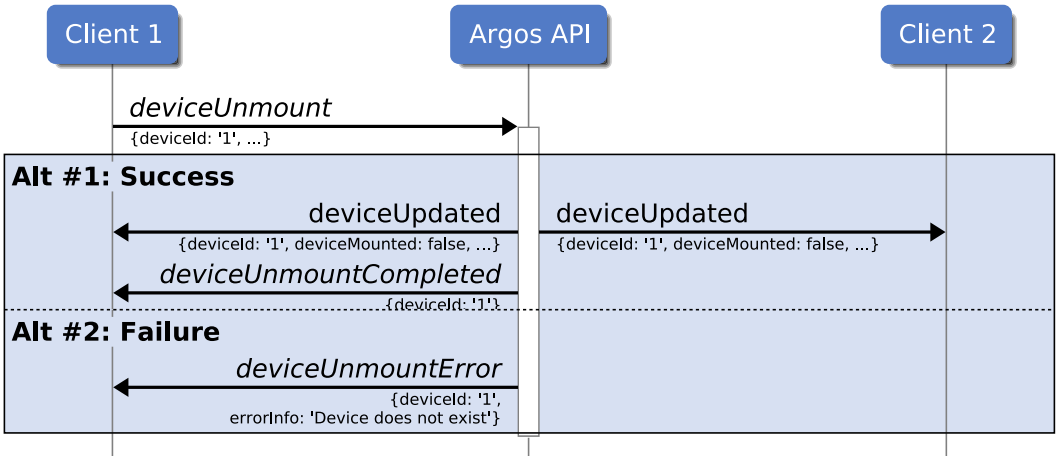

Figure 6.5: Example of deviceUnmount message communication between the client and the ARGOS API.

mation about the device that has been unmounted. In case an error occurs during the «deviceUnmount» a «deviceUnmountError» message will be sent as response.

A child device of a composite device may also be unmounted. This can be used to disable only this child while keeping the rest of the composite device active. A composite device is created with all its children mounted. Children may be unmounted at any time. Mounting/unmounting the parent device does not affect the children's mount status.

Request: deviceUnmount
The «deviceUnmount» request command will set the device in a state where it stops any connection to the physical device and first try to reconnect to the device if a new «deviceMount» is sent.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

When the «deviceUnmount» request has set the state and any previous created connections to the physical devices is stopped a «deviceUpdated» message will be pushed with information about e.g. the deviceMounted state and the deviceState.

Response: deviceUnmountCompleted
When a «deviceUnmount» request command is successfully registered it will respond with a «deviceUnmountCompleted» response and if an error occurs a «deviceUnmountError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/DeviceDeviceIdError.json`

6.6 deviceLocationChange

Pattern: Request/Response

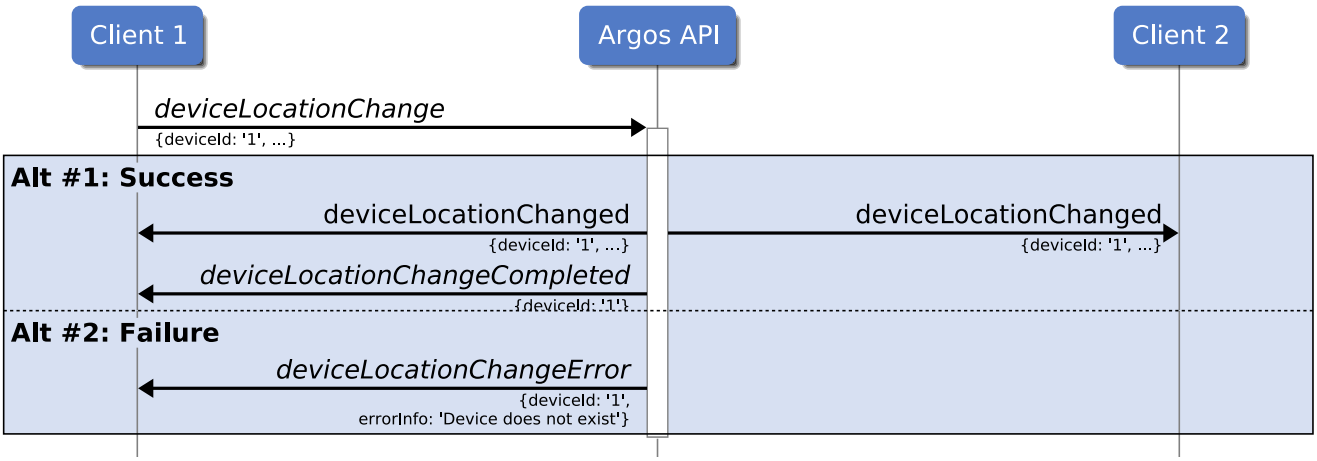

Figure 6.6: Example of deviceLocationChange message communication between the client and the ARGOS API.

When a device needs to be geographically set or updated to a physical geo-location and or heading a «deviceLocationChange» command must be used and a response with «deviceLocationChangeCompleted» will be send back to show that the device has been relocated. The operation causes a push message with a «deviceLocation-Changed» to be sent with information about the device that has been relocated. In case an error occurs during the «deviceLocationChange» a «deviceLocationChangeError» message will be sent as response. This command can also be used on devices that are included in composite devices.

Request: deviceLocationChange
The «deviceLocationChange» request command will set the device location to a new geo-location.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceLocationChange.json`

Response: deviceLocationChangeCompleted
When a «deviceChange» request command is successfully registered it will respond with a «deviceLocationChangeCompleted» response and if an error occurs a «device-LocationChangeError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/DeviceDeviceIdError.json`

6.7 deviceMiscellaneousChange

Pattern: Request/Response

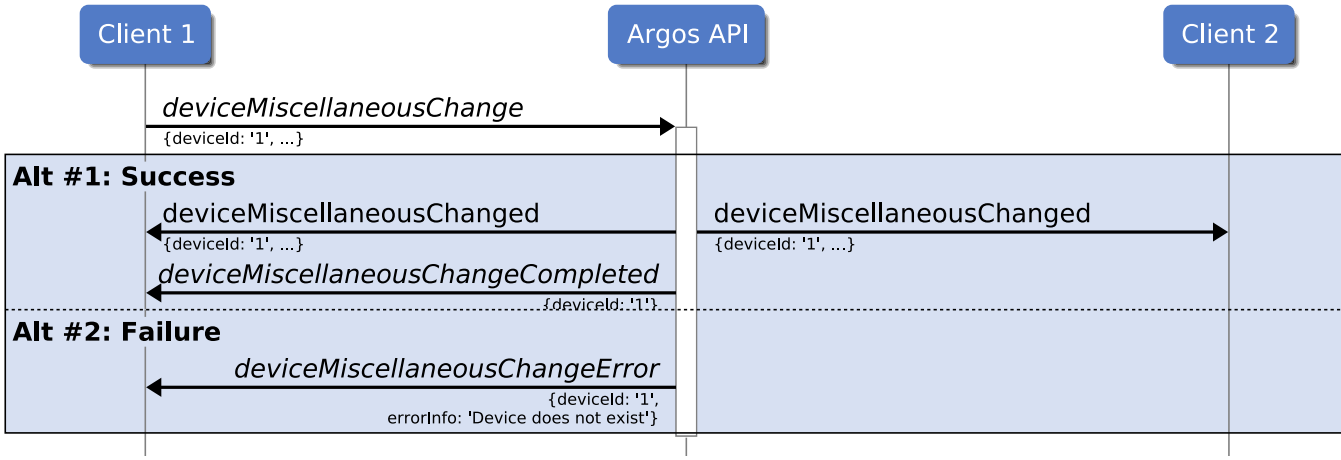

Figure 6.7: Example of deviceMiscellaneousChange message communication between the client and the ARGOS API.

When a device needs to change one of the miscellaneous attributes a «deviceMiscellaneousChange» command must be used and a response with «deviceMiscellaneousChangeCompleted» will be send back to show that the device has changed the miscellaneous attribute. The operation causes a push message with a «deviceMiscellaneousChanged» to be sent with information about the device that has changed miscellaneous data. In case an error occurs during the «deviceMiscellaneousChange» a «deviceMiscellaneousChangeError» message will be sent as response. This command can also be used on devices that are included in composite devices.

Request: deviceMiscellaneousChange
The «deviceMiscellaneousChange» request command will set the device miscellaneous data.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceMiscellaneousInfo.json`

Refer to the descriptions in the schema files for information on the different miscellaneous data. «deviceMiscellaneousUpdate» schema is generated from «deviceMiscellaneous»/«deviceMiscellaneousUpdateInfo» schema. All properties in the latter or included in the former, except those with the comment tag "read-only".

Response: deviceMiscellaneousChangeCompleted
When a «deviceMiscellaneousChange» request command is successfully registered it will respond with a «deviceMiscellaneousChangeCompleted» response and if an error occurs a «deviceMiscellaneousChangeError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or

`schema/messages/DeviceDeviceIdError.json`

6.8 deviceGetLocations

Pattern: Request/Response

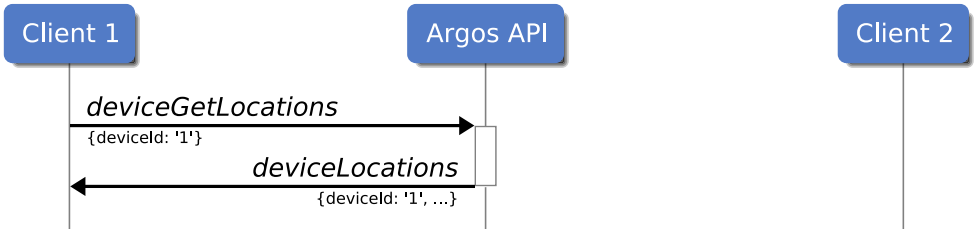

Figure 6.8: Example of deviceGetLocations message communication between the client and the ARGOS API.

If the full list of all locations for a single device is needed the «deviceGetLocations» command must be sent and a response with «deviceLocations» will be send back with the current list of all locations for that device. This command can also be used for devices included in a composite device.

Request: deviceGetLocations
It is possible to send a request to receive the actual list added devices.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

Response: deviceLocations
The response will be a list of all the current locations for the requested device and if no locations is added an empty array will be returned.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceLocationList.json`

6.9 deviceGetMiscellaneous

Pattern: Request/Response

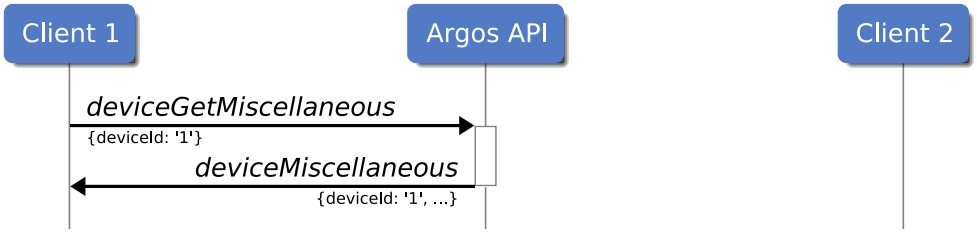

Figure 6.9: Example of deviceGetMiscellaneous message communication between the client and the ARGOS API.

If the full list of all miscellaneous data for a single device is needed the «deviceGet-Miscellaneous» command must be sent and a response with «deviceMiscellaneous» will be send back with the current list of all miscellaneous data for that device. This command can also be used for devices included in a composite device.

Request: deviceGetMiscellaneous
It is possible to send a request to receive the actual list added devices.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

Response: deviceMiscellaneous
The response will be a list of all the currently miscellaneous data for the requested device and if no miscellaneous data is added an empty array will be returned.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceMiscellaneousList.json`

Miscellaneous types not included in the list above can occur and should be ignored and should not be modified. The types are for internal use only and related to the MyDefence IRIS application.

6.10 deviceCalibrationStart

Pattern: Request/Response

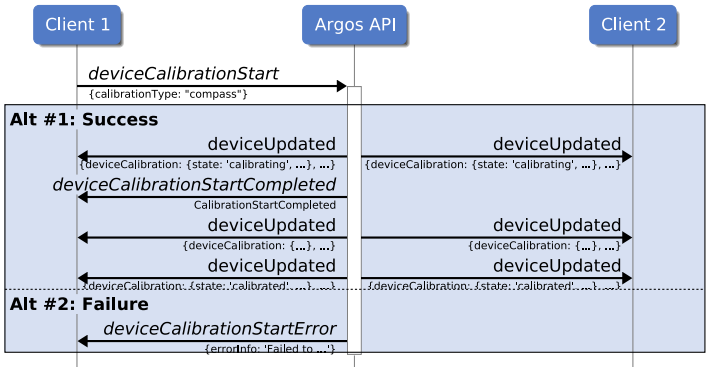

Figure 6.10: Example of deviceCalibrationStart message communication between the client and the ARGOS API.

When using the «deviceCalibrationStart» command, a response «deviceCalibrationStart-Completed» message will be sent back as confirm. The status of the calibration is signalled using «deviceUpdated» messages (section 5.3). In case an error occurs during «deviceCalibrationStart» processing, the «deviceCalibrationStartError» message will be sent as response.

Devices supporting calibration will have a deviceCalibration property in their device data.

For composite devices, deviceId in the «deviceCalibrationStart» command must be the deviceId of a child device supporting calibration.

Request: deviceCalibrationStart
It is possible to send a request to start the calibration process for a device.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceCalibrationStart.json` or

Response: deviceCalibrationStartCompleted
When a «deviceCalibrationStart» request command is successfully registered it will respond with a «deviceCalibrationStartCompleted» response and if an error occurs a «deviceCalibrationStartError» response.

`schema/messages/DeviceDeviceId.json` or
`schema/messages/DeviceDeviceIdError.json` or

7. Receiving uthreats

The uthreat interface delivers a unified threat interface across the system irrespective of the kind of sensor device(s) delivering the raw detection information to ARGOS. Furthermore, the uthreat interface can be configured to deliver various levels of fused data. The levels are:

none No fusing of threats. Threats are delivered per sensor and intended for 3rd party integrators wishing to do their own fusing of threats.

rf Fusing of threats for MyDefence RF sensors only. Threats from other sources like radars are delivered separately on a per device level.

all Fusing of data from sensors. Where possible, threats from different kinds of sensors (e.g. radar and rf) will be fused and delivered as a single uthreat series.

For all fusion levels, the uthreat interface is the same, and delivers threat information in four levels according to the following scheme:

1. Location
2. Direction
3. Zone
4. Presence

When one of the fusing levels (rf, all) have been configured, a given uthreat series related to a particular physical threat may go up or down through these levels in its lifetime, all depending on the collective information available to ARGOS at any given time. Going up in level can be exemplified by multiple directions being combined via triangulation to form a location. Similarly going down in level could be because direction data becomes to uncertain and therefore a presence level is emitted.

To facilitate the functionality of the uthreat interface the mission center has to be configured (described in chapter 17 on page 97).

7.1 uthreat

Pattern: Push

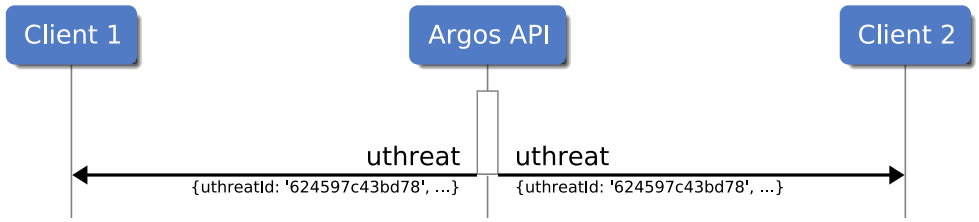

Figure 7.1: Example of uthreat message communication from ARGOS API to a connected client.

When a physical threat is detected by one or more components in the system «uthreat» messages starts being sent. Every time an update to the related physical threat is made a «uthreat» message is sent. In the first «uthreat» message the createdTimeStamp and updatedTimeStamp fields are identical and stoppedTimestamp is not present. In the following «uthreat» messages where updates have been made, the updatedTimeStamp is updated accordingly and the stoppedTimestamp is not present. When the physical threat is not detected by any component in the system anymore the «uthreat» is stopped and the stoppedTimestamp field is set.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Uthreat.json`

7.2 uthreatConfigure

Pattern: Request/Response

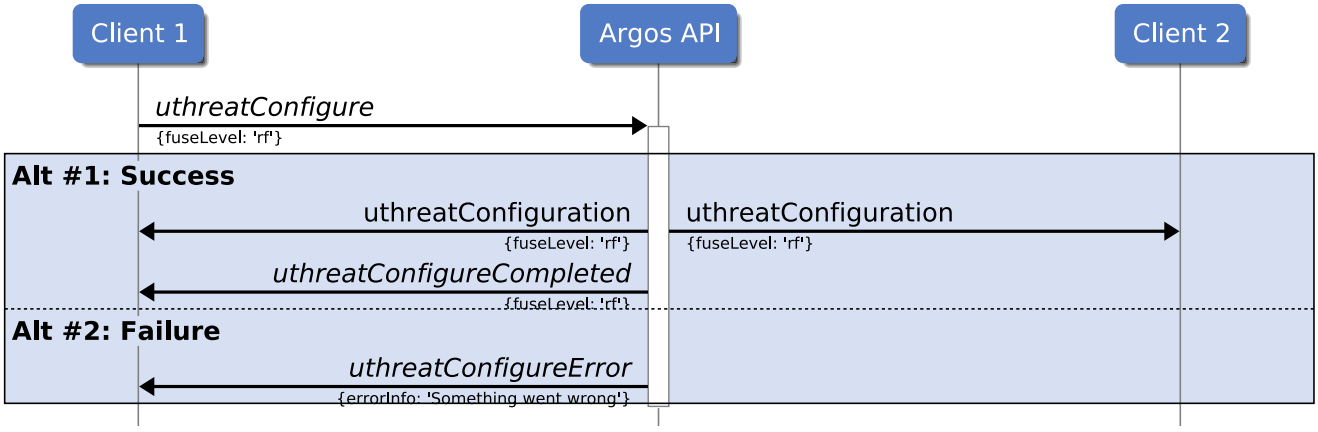

Figure 7.2: Example of uthreatConfigure message communication between the client and the ARGOS API.

Configuration of the uthreat layer can be set via the «uthreatConfigure» command. ARGOS sends a response with «uthreatConfigureCompleted» back to the requesting client. The operation causes a «uthreatConfiguration» push message with the current

configuration to be sent. In case an error occurs during the «uthreatConfigure» a «uthreatConfigureError» message will be sent as response. Sending uthreatConfigure may result in current threats being reset.

Request: uthreatConfigure
A «uthreatConfigure» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatConfig.json`

Response: uthreatConfigureCompleted
When a «uthreatConfigure» request command is successfully registered it will respond with a «uthreatConfigureCompleted» response and if an error occurs a «uthreatConfigureCompletedError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatConfig.json` or
`schema/messages/Error.json`

7.3 uthreatConfiguration

Pattern: Push

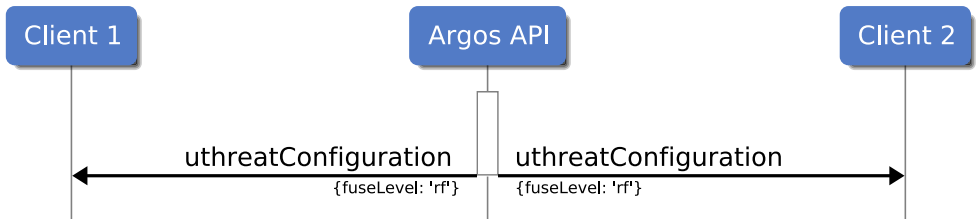

Figure 7.3: Example of uthreatConfiguration message communication from ARGOS API to a connected client.

When the uthreat configuration is updated by a «uthreatConfigure» a «uthreatConfiguration» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatConfig.json`

7.4 uthreatGetConfiguration

Pattern: Request/Response

To get the current uthreat configuration the following message must be sent. The device will then return the current uthreat configuration.

Request: uthreatGetConfiguration
It is possible to send a request to get the current uthreat configuration.

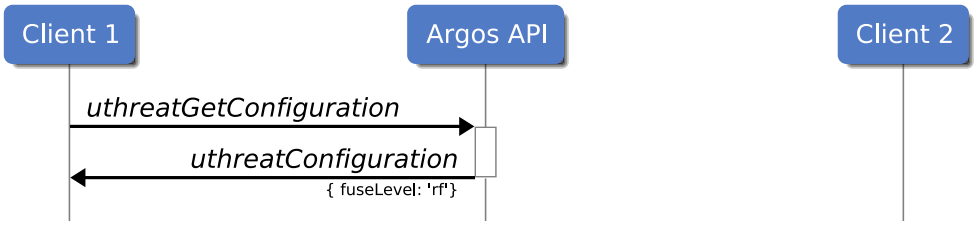

Figure 7.4: Example of uthreatGetConfiguration message communication between the client and the ARGOS API.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: uthreatConfiguration
When an «uthreatGetConfiguration»» request command is successfully registered it will respond with an «uthreatConfiguration» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatConfig.json`

7.5 uthreatGetList

Pattern: Request/Response

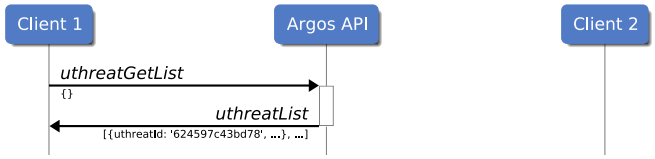

Figure 7.5: Example of uthreatGetList message communication between the client and the ARGOS API.

When using the «uthreatGetList» command a response with «uthreatList» will be sent back as response.

Request: uthreatGetList
Use this to request a list of all active threats.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json`

Response: uthreatList
When «uthreatGetList» request command is successfully registered it will respond with the «uthreatList» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/UthreatList.json`

7.6 uthreatList

Pattern: Push

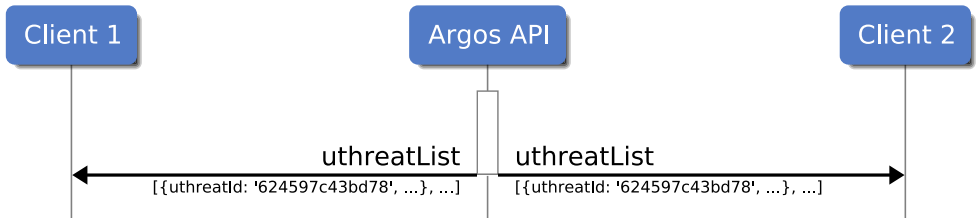

Figure 7.6: Example of uthreatList message communication from ARGOS API to a connected client when a new alarm zone has been created.

Argos may push a complete threat list if needed to synchronize state between Argos and client. Usually only «uthreat» events are pushed when starting or stopping threats, but a client must monitor this also.

7.7 uthreatSetMute

Pattern: Request/Response

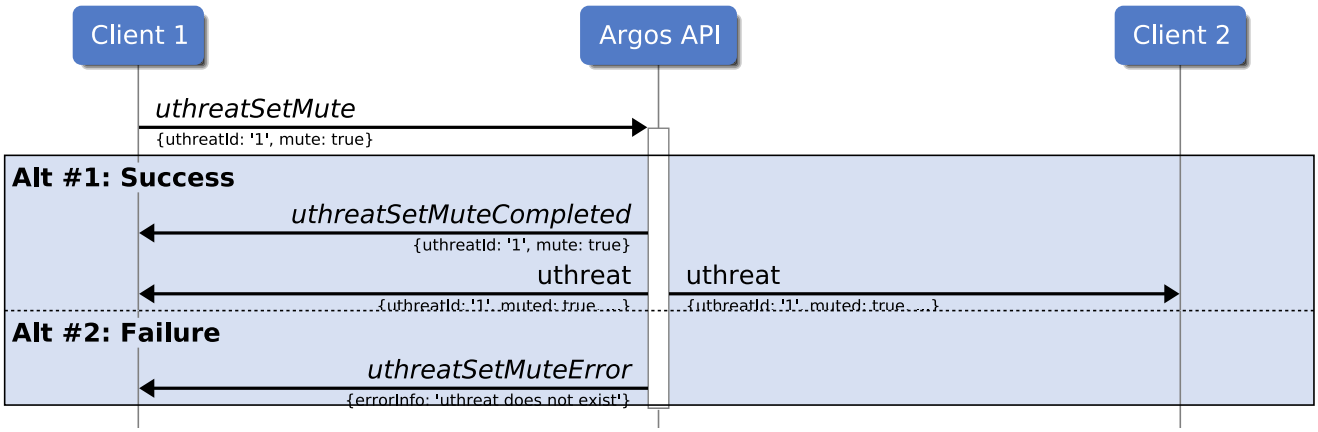

Figure 7.7: Example of uthreatSetMute message communication between the client and the ARGOS API.

Mute/unmute of a uthreat can be requested via a «uthreatSetMute» command. ARGOS sends a response with «uthreatSetMuteCompleted» back to the requesting client. The mute setting is reflected in subsequent «uthreat» push messages for the given uthreatId. In case an error occurs during the «uthreatSetMute» a «uthreatSetMuteError» message will be sent as response.

Request: uthreatSetMute
A «uthreatSetMute» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatSetMute.json`

Response: uthreatSetMuteCompleted
When a «uthreatSetMute» request command is successfully registered it will respond with a «uthreatSetMuteCompleted» response and if an error occurs a «uthreatSetMuteError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatSetMute.json` or
`schema/messages/Error.json`

7.8 uthreatGetHistory

Pattern: Request/Response

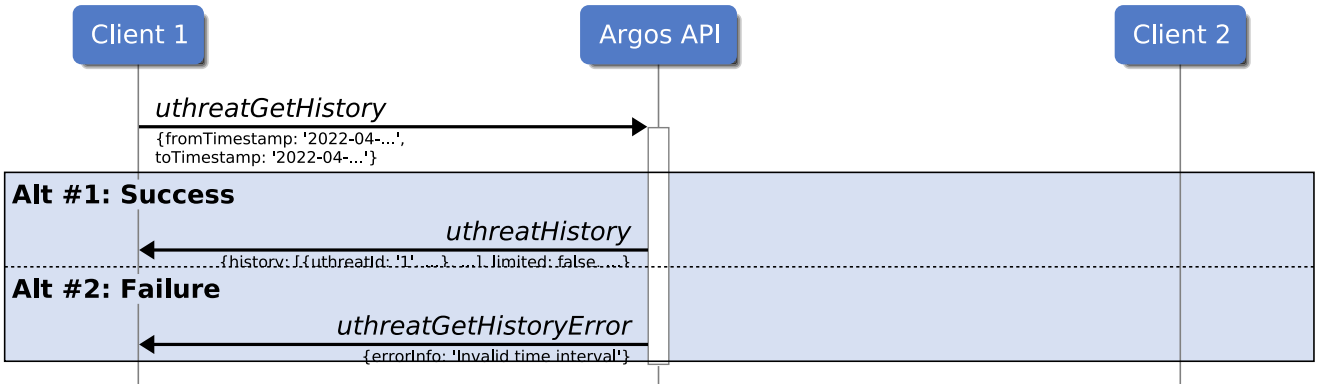

Figure 7.8: Example of uthreatGetHistory message communication between the client and the ARGOS API.

An overview of the uthreats that were registered in a given time frame can be requested via a «uthreatGetHistory» command. ARGOS sends a response with «uthreatHistory» back to the requesting client.

Request: uthreatGetHistory
A «uthreatGetHistory» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/TimeInterval.json`

Response: uthreatHistory
When a «uthreatGetHistory» request command is successfully registered it will respond with a «uthreatHistory» response and if an error occurs a «uthreatGetHistoryError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatHistory.json` or
`schema/messages/Error.json`

7.9 uthreatGetHistoryData

Pattern: Request/Response

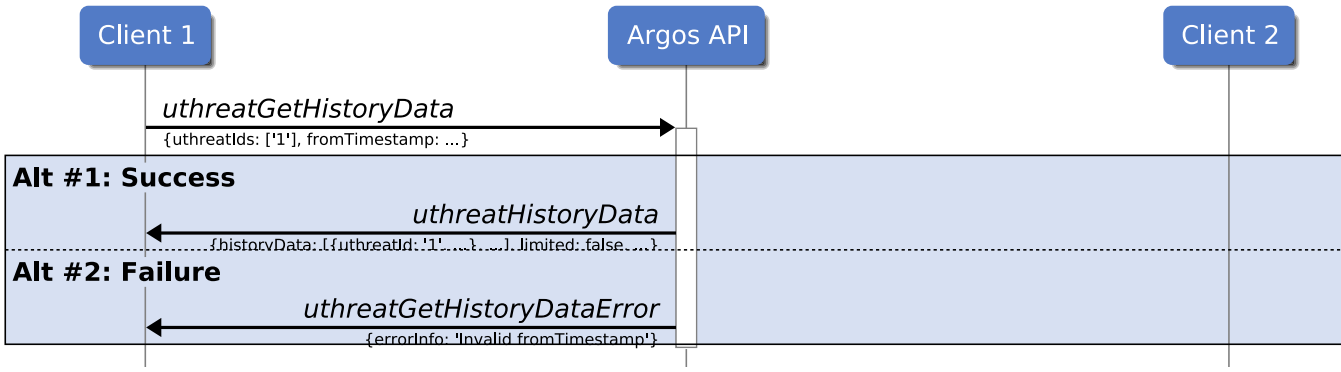

Figure 7.9: Example of uthreatGetHistoryData message communication between the client and the ARGOS API.

A list of all the data for a given set of uthreats can be requested via a «uthreatGetHistoryData» command. ARGOS sends a response with «uthreatHistoryData» back to the requesting client.

Request: uthreatGetHistoryData
A «uthreatGetHistoryData» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatGetHistoryData.json`

Response: uthreatHistoryData
When a «uthreatGetHistoryData» request command is successfully registered it will respond with a «uthreatHistoryData» response and if an error occurs a «uthreatGetHistoryDataError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/UthreatHistoryData.json` or
`schema/messages/Error.json`

8. Receiving alarms

There are 3 types of alarm messages «alarmStarted», «alarmStopped» and «alarmUpdated».

8.1 alarmStarted

Pattern: Push

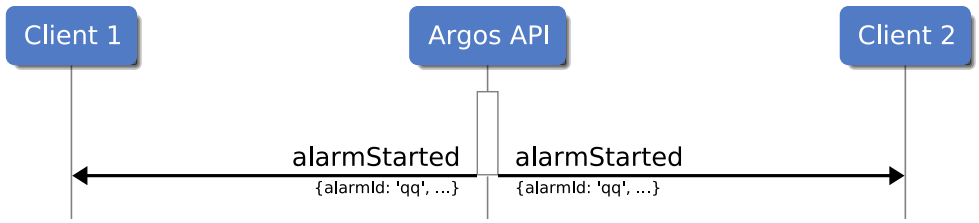

Figure 8.1: Example of alarmStarted message communication from ARGOS API to a connected client.

When a alarm is detected for the first time a «alarmStarted» message is sent. The stopTimeStamp will be null.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Alarm.json`

8.2 alarmStopped

Pattern: Push

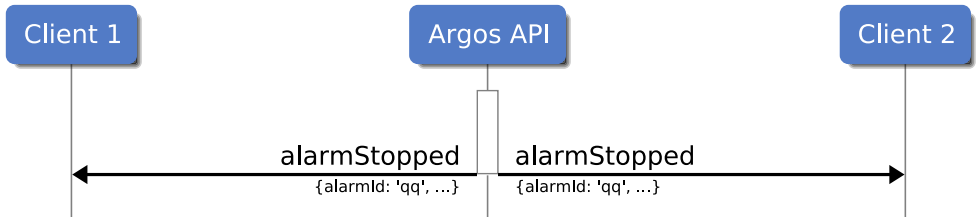

Figure 8.2: Example of alarmStopped message communication from ARGOS API to a connected client.

When a alarm is detected for the last time a «alarmStopped» message is sent. The stopTimeStamp will be filled with a value.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Alarm.json`

8.3 alarmUpdated

Pattern: Push

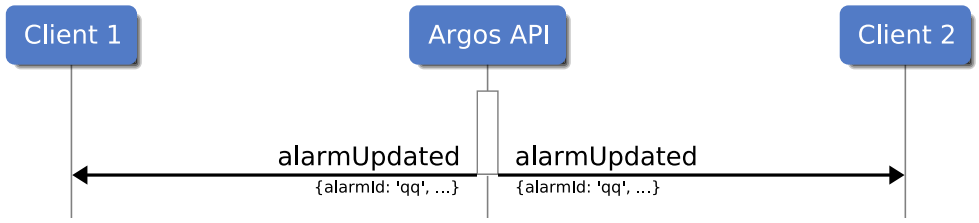

Figure 8.3: Example of alarmUpdated message communication from ARGOS API to a connected client.

When a alarm is updated but not stopped a «alarmUpdated» message is sent. The createdTimeStamp and the updatedTimeStamp will be different.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Alarm.json`

9. Alarm Zones

9.1 alarmZoneAdd

Pattern: Request/Response

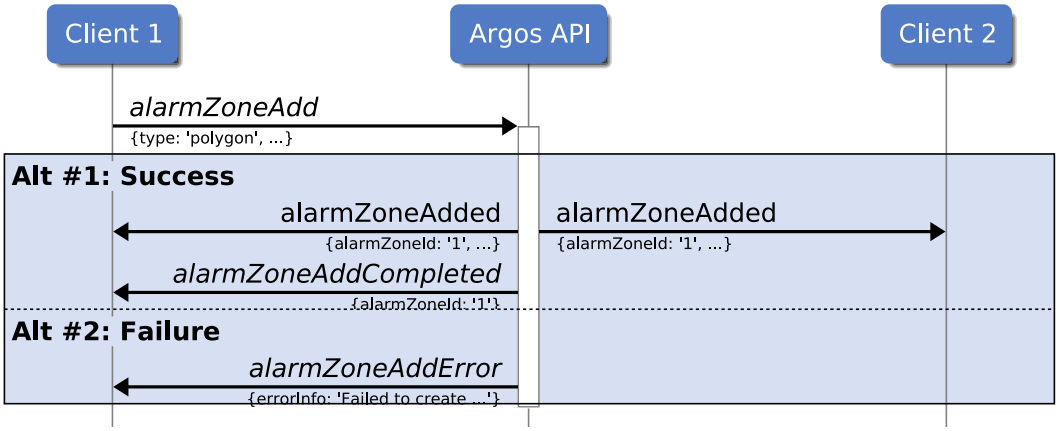

Figure 9.1: Example of alarmZoneAdd message communication between the client and the ARGOS API.

An alarm zone can be created with the «alarmZoneAdd» request. When using the «alarmZoneAdd» command a response with «alarmZoneAddCompleted» will be sent back to show that the alarm zone has been created. The operation causes a push message with a «alarmZoneAdded» to be sent with information about the alarm zone that has been created. In case an error occurs during the «alarmZoneAdd» an «alarmZoneAddError» message will be sent as response.

Request: alarmZoneAdd
The «alarmZoneAdd» request will create the alarm zone.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneCreate.json`

When the «alarmZoneAdd» request has processed correctly an «alarmZoneAdded» message will be pushed with information about the created alarm zone.

Response: alarmZoneAddCompleted
When an «alarmZoneAdd» request command is successfully registered it will respond with an «alarmZoneAddCompleted» response and if an error occurs an «alarmZoneCreateError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneId.json` or
`schema/messages/Error.json`

9.2 alarmZoneUpdate

Pattern: Request/Response

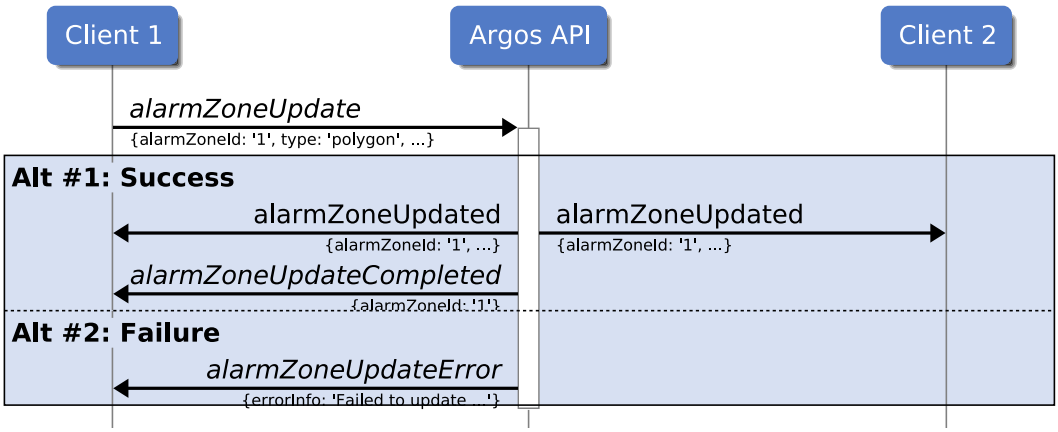

Figure 9.2: Example of alarmZoneUpdate message communication between the client and the ARGOS API.

An alarm zone can be updated with the «alarmZoneUpdate» request. When using the «alarmZoneUpdate» command a response with «alarmZoneUpdateCompleted» will be sent back to show that the alarm zone has been updated. The operation causes a push message with a «alarmZoneUpdated» to be sent with information about the alarm zone that has been updated. In case an error occurs during the «alarmZoneUpdate» an «alarmZoneUpdateError» message will be sent as response.

Request: alarmZoneUpdate
The «alarmZoneUpdate» request will update the alarm zone.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZone.json`

When the «alarmZoneUpdate» request has processed correctly an «alarmZoneUpdated» message will be pushed with information about the updated alarm zone.

Response: alarmZoneUpdateCompleted
When an «alarmZoneUpdate» request command is successfully registered it will respond with an «alarmZoneUpdateCompleted» response and if an error occurs an «alarmZoneUpdateError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneId.json` or
`schema/messages/Error.json`

9.3 alarmZoneRemove

Pattern: Request/Response

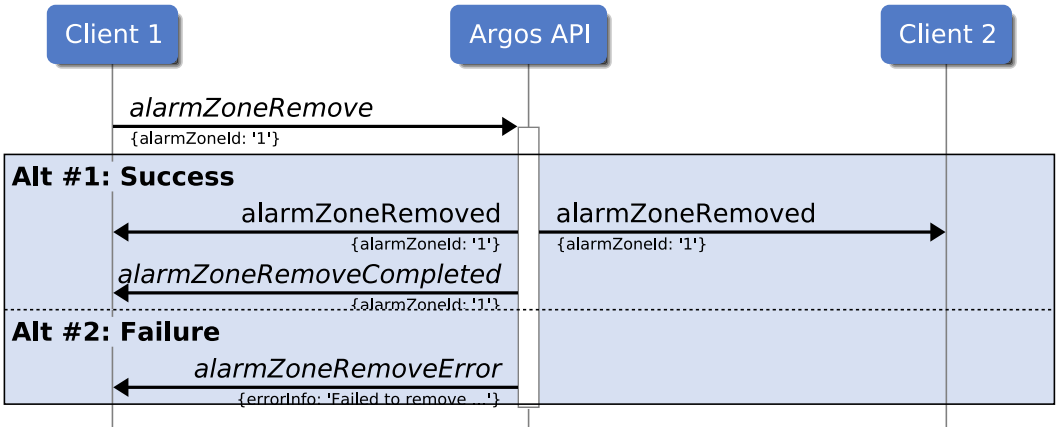

Figure 9.3: Example of alarmZoneRemove message communication between the client and the ARGOS API.

An alarm zone can be removed with the «alarmZoneRemove» request. When using the «alarmZoneRemove» command a response with «alarmZoneRemoveCompleted» will be sent back to show that the alarm zone has been removed. The operation causes a push message with a «alarmZoneRemoved» to be sent with information about the alarm zone that has been removed. In case an error occurs during the «alarmZoneRemove» an «alarmZoneRemoveError» message will be sent as response.

Request: alarmZoneRemove
The «alarmZoneRemove» request will remove the alarm zone.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneId.json`

When the «alarmZoneRemove» request has processed correctly an «alarmZoneRemoved» message will be pushed with information about the removed alarm zone.

Response: alarmZoneRemoveCompleted
When an «alarmZoneRemove» request command is successfully registered it will respond with an «alarmZoneRemoveCompleted» response and if an error occurs an «alarmZoneRemoveError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneId.json` or
`schema/messages/Error.json`

9.4 alarmZoneAdded

Pattern: Push

When a new alarm zone is created an «alarmZoneAdded» message is pushed to all connected clients.

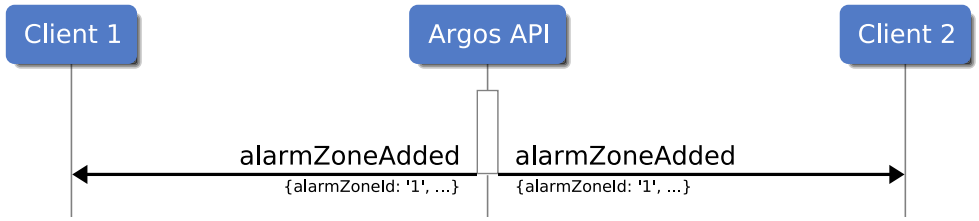

Figure 9.4: Example of alarmZoneAdded message communication from ARGOS API to a connected client when a new alarm zone has been created.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZone.json`

9.5 alarmZoneUpdated

Pattern: Push

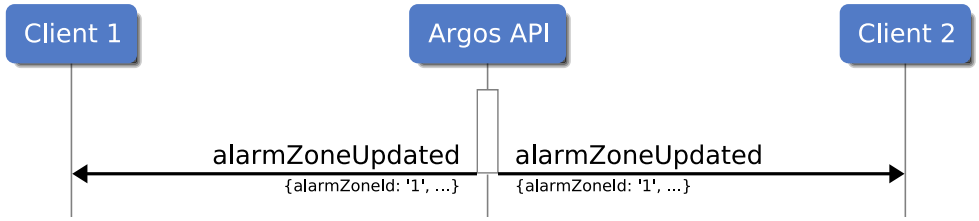

Figure 9.5: Example of alarmZoneUpdated message communication from ARGOS API to a connected client when an alarm zone has been updated.

When an alarm zone is updated an «alarmZoneUpdated» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZone.json`

9.6 alarmZoneRemoved

Pattern: Push

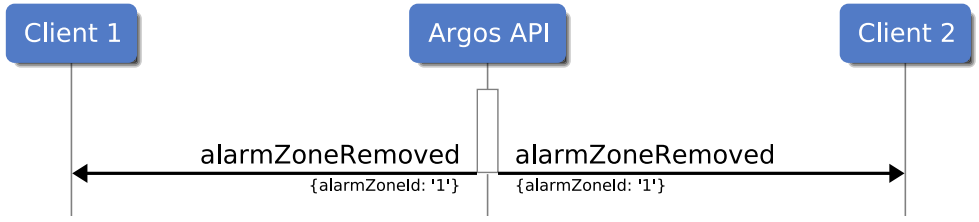

Figure 9.6: Example of alarmZoneRemoved message communication from ARGOS API to a connected client when an alarm zone has been removed.

When an alarm zone is removed an «alarmZoneRemoved» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneId.json`

9.7 alarmZoneGetList

Pattern: Request/Response

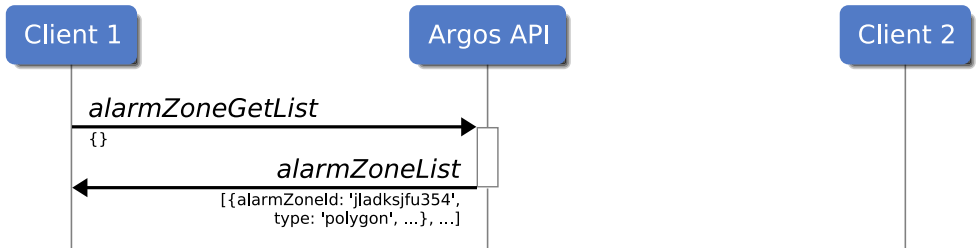

Figure 9.7: Example of alarmZoneGetList message communication between the client and the ARGOS API.

A list of the currently existing alarm zones can be requested via an «alarmZoneGetList» command. ARGOS sends a response with «alarmZoneList» back to the requesting client.

Request: alarmZoneGetList
An «alarmZoneGetList» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: alarmZoneList
When an «alarmZoneGetList» request command is successfully registered it will respond with an «alarmZoneList» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneList.json`

9.8 alarmZoneList

Pattern: Push

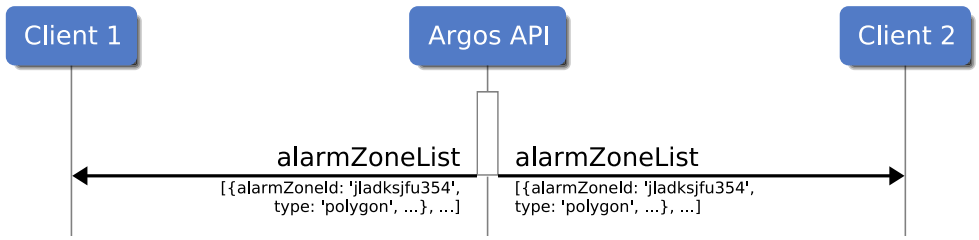

Figure 9.8: Example of alarmZoneList message communication from ARGOS API to a connected client.

During ARGOS startup or if an unforeseen error causes a restart of the internal alarm zone service an «alarmZoneList» message is pushed to all connected clients. The list describes all the currently existing alarm zones. This list can also be requested via «alarmZoneGetList» described in section 9.7 on the preceding page.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/AlarmZoneList.json`

10. Origin Filters

Origin filters are are used with radars to filter tracks from zones with many false (non-drone) tracks. The need for origin filters will depend on the different radar's ability to separate drones from other tracks. It has no effect for RF sensors.

For example cars on a road may generate tracks. In that case put an origin filter zone around the part of the road visible from the radar. Origin filters will only filter out threats, that start (originate) in the specified zone. Threats starting outside a zone and passing through are not affected.

Another special example is that ventilation fans may confuse some radars: They generate false tracks starting at the fan, but the radar may see the track as moving. In that case creating an origin filter, with a distance setting of e.g. some hundred meters, will filter all tracks that started at that fan, also when moving out of the filter zone, unless tha ttrack is more than the distance away from the zone.

Note: If multiple origin filters match a threat, then an arbitrary will be selected for distance measurement, not all. Typically it is not needed to make overlapping origin filter zones.

The API to specify origin filter zones is similar to the alarm zone API, but also supports height information.

10.1 originFilterAdd

Pattern: Request/Response

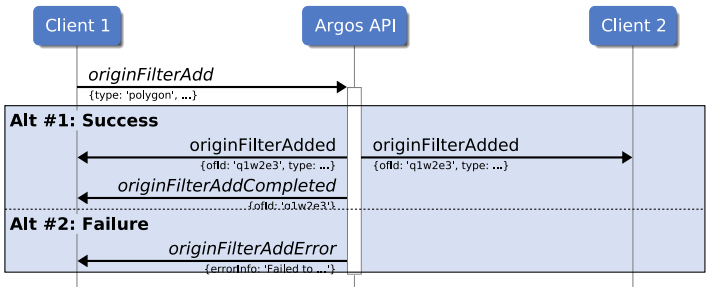

Figure 10.1: Example of originFilterAdd message communication between the client and the ARGOS API.

Use «originFilterAdd» to create and activate an origin filter.

When using the «originFilterAdd» command, a response «originFilterAddCompleted» message will be sent back as confirm. The operation causes a push «originFilterAdded» message to be sent with information about the update. In case an error occurs during «originFilterAdd» processing, the «originFilterAddError» message will be sent

as response.

Request: originFilterAdd

The «originFilterAdd» request creates an origin filter zone.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilter.json`

When the «originFilterAdd» request has processed correctly the «originFilterAdded» message will be pushed with information about the update.

Response: originFilterAddCompleted
When «originFilterAdd» request command is successfully registered it will respond with the «originFilterAddCompleted» response, or if an error occurs the «originFilterAddError» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilterId.json` or
`schema/messages/Error.json`

10.2 originFilterUpdate

Pattern: Request/Response

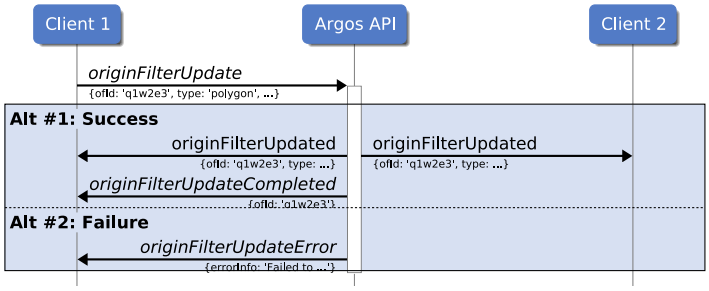

Figure 10.2: Example of originFilterUpdate message communication between the client and the ARGOS API.

The filters can be modified using «originFilterUpdate»

When using the «originFilterUpdate» command, a response «originFilterUpdateCompleted» message will be sent back as confirm. The operation causes a push «originFilterUpdated» message to be sent with information about the update. In case an error occurs during «originFilterUpdate» processing, the «originFilterUpdateError» message will be sent as response.

Request: originFilterUpdate

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilter.json`

When the «originFilterUpdate» request has processed correctly the «originFilterUpdated» message will be pushed with information about the update.

Response: originFilterUpdateCompleted
When «originFilterUpdate» request command is successfully registered it will respond with the «originFilterUpdateCompleted» response, or if an error occurs the «originFilterUpdateError» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilterId.json` or
`schema/messages/Error.json`

10.3 originFilterRemove

Pattern: Request/Response

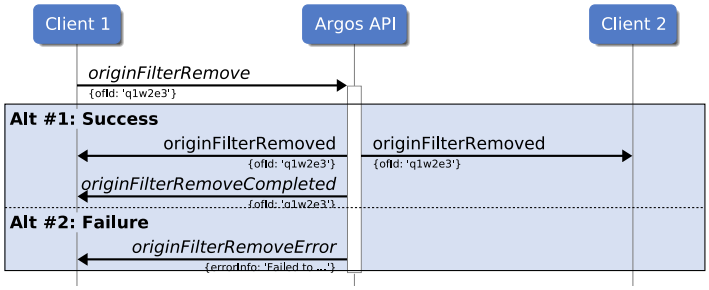

Figure 10.3: Example of originFilterRemove message communication between the client and the ARGOS API.

The filters can be removed using «originFilterRemove»

When using the «originFilterRemove» command, a response «originFilterRemoveCompleted» message will be sent back as confirm. The operation causes a push «originFilterRemoved» message to be sent with information about the update. In case an error occurs during «originFilterRemove» processing, the «originFilterRemoveError» message will be sent as response.

Request: originFilterRemove

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilterId.json`

When the «originFilterRemove» request has processed correctly the «originFilterRemoved» message will be pushed with information about the update.

Response: originFilterRemoveCompleted

When «originFilterRemove» request command is successfully registered it will respond with the «originFilterRemoveCompleted» response, or if an error occurs the «originFilterRemoveError» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilterId.json` or
`schema/messages/Error.json`

10.4 originFilterAdded

Pattern: Push

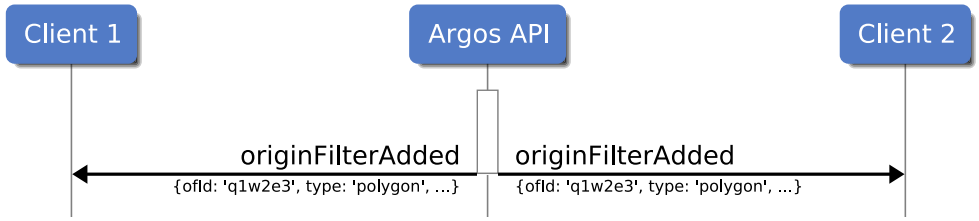

Figure 10.4: Example of originFilterAdded message communication from ARGOS API to a connected client when a new alarm zone has been created.

When a «originFilterAdd» has been processed, a «originFilterAdded» message is pushed to all connected clients.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilter.json`

10.5 originFilterUpdated

Pattern: Push

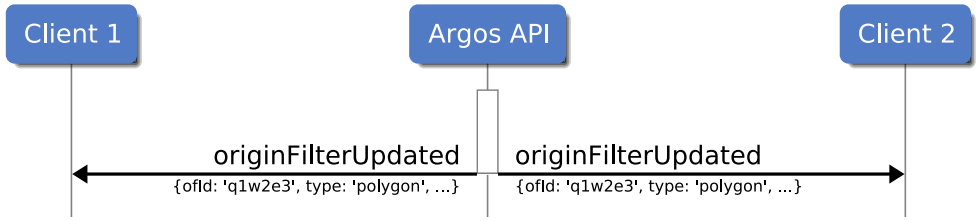

Figure 10.5: Example of originFilterUpdated message communication from ARGOS API to a connected client when a new alarm zone has been created.

When a «originFilterUpdate» has been processed, a «originFilterUpdated» message is pushed to all connected clients.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilter.json`

10.6 originFilterRemoved

Pattern: Push

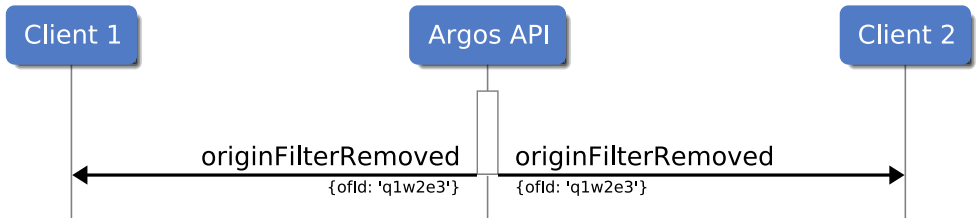

Figure 10.6: Example of originFilterRemoved message communication from ARGOS API to a connected client when a new alarm zone has been created.

When a «originFilterRemove» has been processed, a «originFilterRemoved» message is pushed to all connected clients.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilterId.json`

10.7 originFilterGetList

Pattern: Request/Response

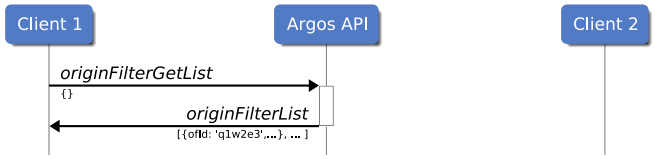

Figure 10.7: Example of originFilterGetList message communication between the client and the ARGOS API.

Use «originFilterGetList» to get a list of all existing filters.

When using the «originFilterGetList» command a response with «originFilterList» will be sent back as response.

Request: originFilterGetList
«originFilterGetList» may be sent at any time.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json`

Response: originFilterList
When «originFilterGetList» request command is successfully registered it will respond with the «originFilterList» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/OriginFilterList.json`

10.8 originFilterList

Pattern: Push

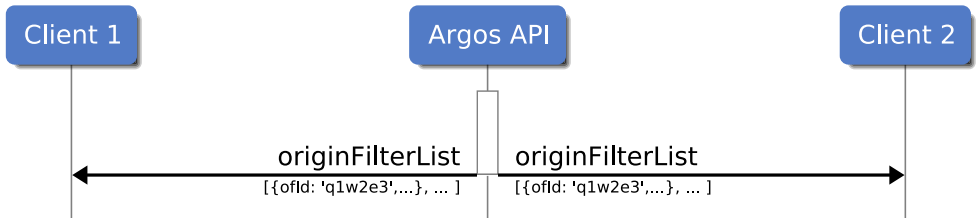

Figure 10.8: Example of originFilterList message communication from ARGOS API to a connected client when a new alarm zone has been created.

During ARGOS startup or if an unforeseen error causes a restart of the internal origin filter service an «originFilterList» message is pushed to all connected clients. The list describes all the currently existing filters. This list can also be requested via «originFilterGetList» described in section 10.7 on the previous page.

11. Threat type handling

When a RF sensor is mounted in the system ARGOS retrieves a list supported threat types from the device. A complete list of supported threat types is maintained in ARGOS. Thus, multiple RF sensors possibly with different firmware version is supported. Each threat type can be muted/unmuted separately.

11.1 threatTypeList

Pattern: Push

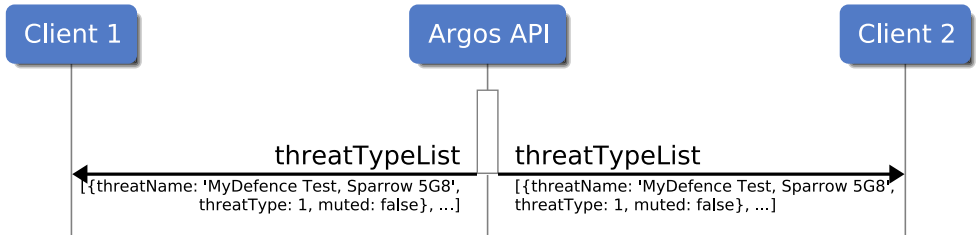

Figure 11.1: Example of threatTypeList message communication from ARGOS API to a connected client when a new device is attached to the network.

When a new MyDefence RF device is mounted the internally maintained list of supported threat types is updated. When the list is updated a «threatTypeList» message is pushed to all clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/ThreatTypeList.json`

11.2 threatTypeGetList

Pattern: Request/Response

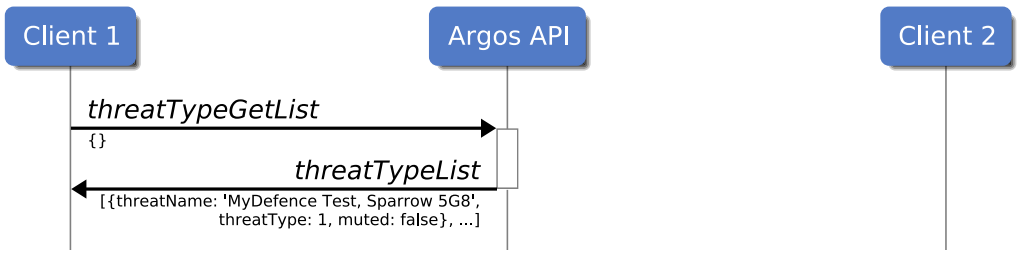

Figure 11.2: Example of threatTypeGetList message communication between the client and the ARGOS API.

A list of the currently supported threat types can be requested via a «threatTypeGetList» command. ARGOS sends a response with «threatTypeList» back to the re-

questing client.

Request: threatTypeGetList
A «threatTypeGetList» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: threatTypeList
When a «threatTypeGetList» request command is successfully registered it will respond with a «threatTypeList» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/ThreatTypeList.json`

11.3 threatTypeMute

Pattern: Request/Response

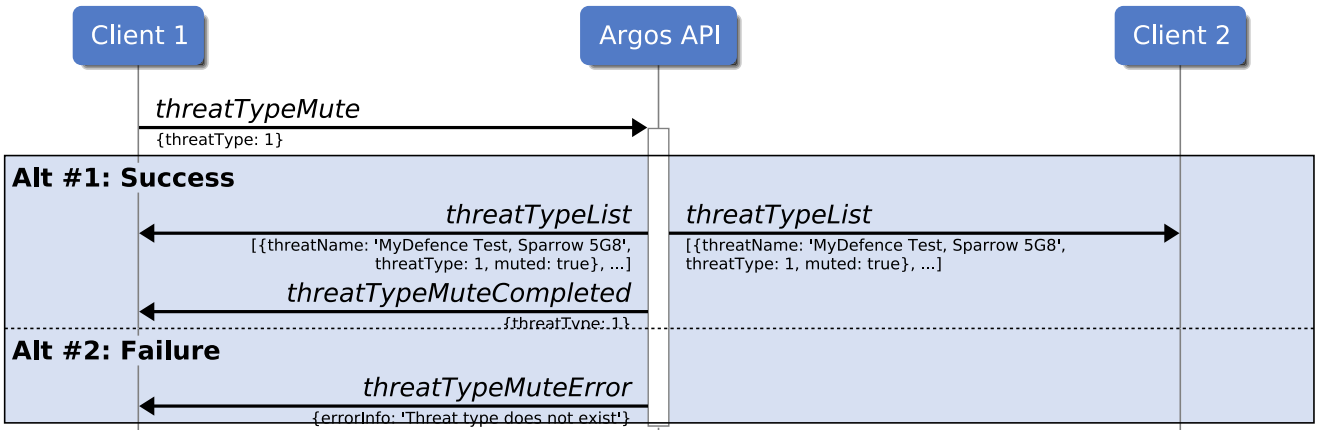

Figure 11.3: Example of threatTypeMute message communication between the client and the ARGOS API.

Mute of a threat type can be requested via a «threatTypeMute» command. ARGOS sends a response with «threatTypeMuteCompleted» back to the requesting client. The change is pushed to all clients with a «threatTypeList». The mute setting is reflected in subsequent «uthreat» push messages for the given threat type. In case an error occurs during the «threatTypeMute» a «threatTypeMuteError» message will be sent as response.

Request: threatTypeMute
A «threatTypeMute» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/ThreatType.json`

Response: threatTypeMuteCompleted

When a «threatTypeMute» request command is successfully registered it will respond with a «threatTypeMuteCompleted» response and if an error occurs a «threatTypeMuteCompletedError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/ThreatType.json` or
`schema/messages/Error.json`

11.4 threatTypeUnMute

Pattern: Request/Response

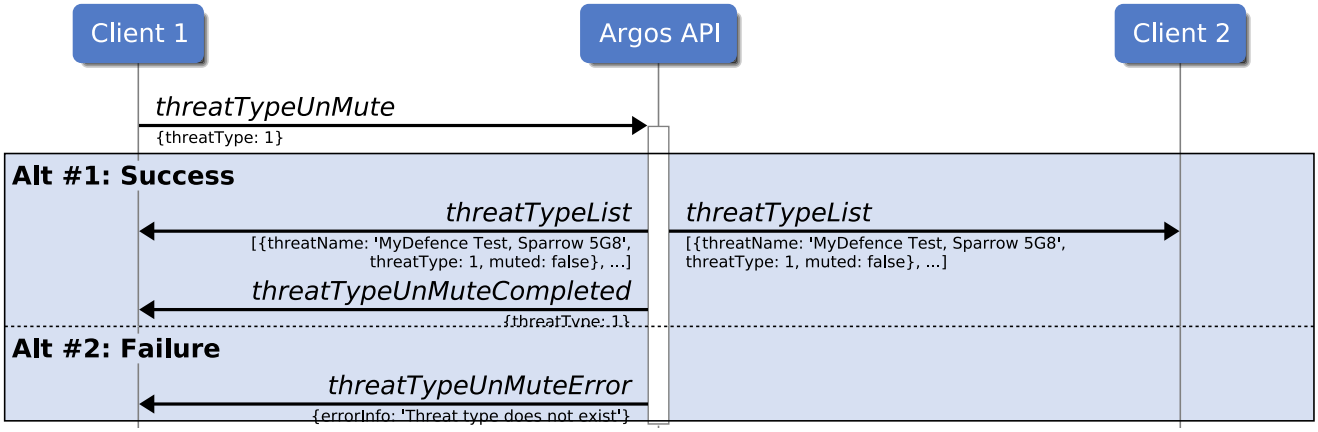

Figure 11.4: Example of threatTypeUnMute message communication between the client and the ARGOS API.

UnMute of a threat type can be requested via a «threatTypeUnMute» command. ARGOS sends a response with «threatTypeUnMuteCompleted» back to the requesting client. The change is pushed to all clients with a «threatTypeList». The Unmute setting is reflected in subsequent «uthreat» push messages for the given threat type. In case an error occurs during the «threatTypeUnMute» a «threatTypeUnMuteError» message will be sent as response.

Request: threatTypeUnMute
A «threatTypeUnMute» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/ThreatType.json`

Response: threatTypeUnMuteCompleted
When a «threatTypeUnMute» request command is successfully registered it will respond with a «threatTypeUnMuteCompleted» response and if an error occurs a «threatTypeUnMuteCompletedError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/ThreatType.json` or

schema/messages/Error.json

12. Communicating with an ECM devices

When an ECM device is mounted in ARGOS and a connection is made between ARGOS and the device it is possible to use the ECM capability. Currently only 'Manual' mode is supported. The frequency band to use is configured via the miscellaneous data for the ECM device. See section 6.7 on page 35 for details regarding this. The overall flow of controlling an ECM device is shown in fig. 12.1.

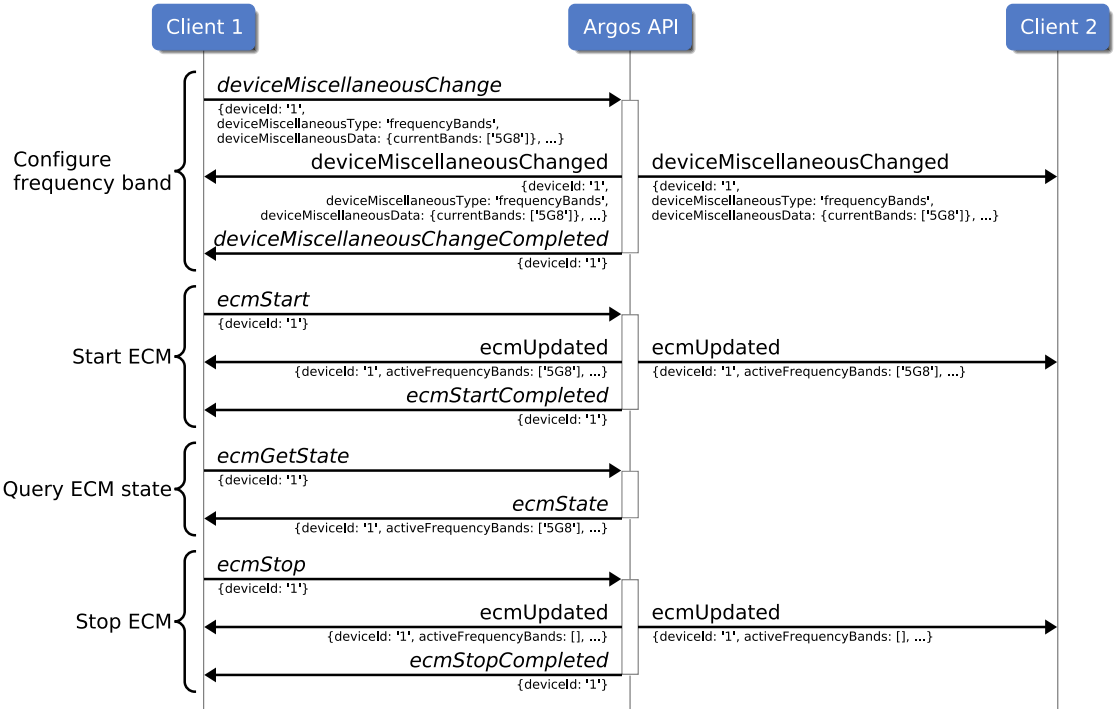

Figure 12.1: Example ECM control communication between the client and the ARGOS API.

12.1 ecmStart

Pattern: Request/Response

To start an ECM on a device the following message must be sent. The device will then start the ECM and run for up to 5 minutes or until it is stopped. Only available in manual mode.

Request: ecmStart
It is possible to send a request to start an effector jamming on a specific frequency band.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

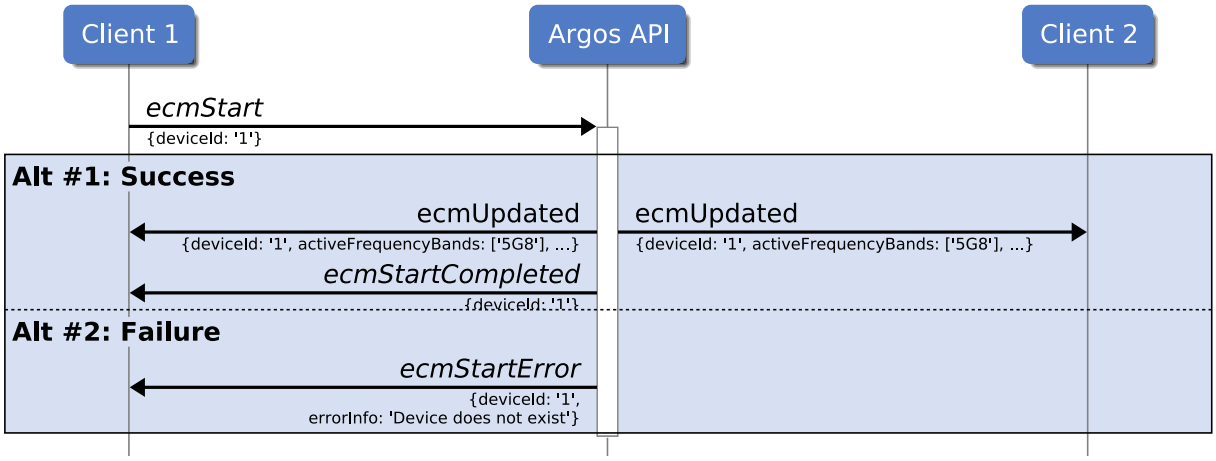

Figure 12.2: Example of ecmStart message communication between the client and the ARGOS API.

Response: ecmStartCompleted
When a «ecmStart» request command is successfully registered it will respond with a «ecmStartCompleted» response and if an error occurs a «ecmStartError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/DeviceDeviceIdError.json`

12.2 ecmStop

Pattern: Request/Response

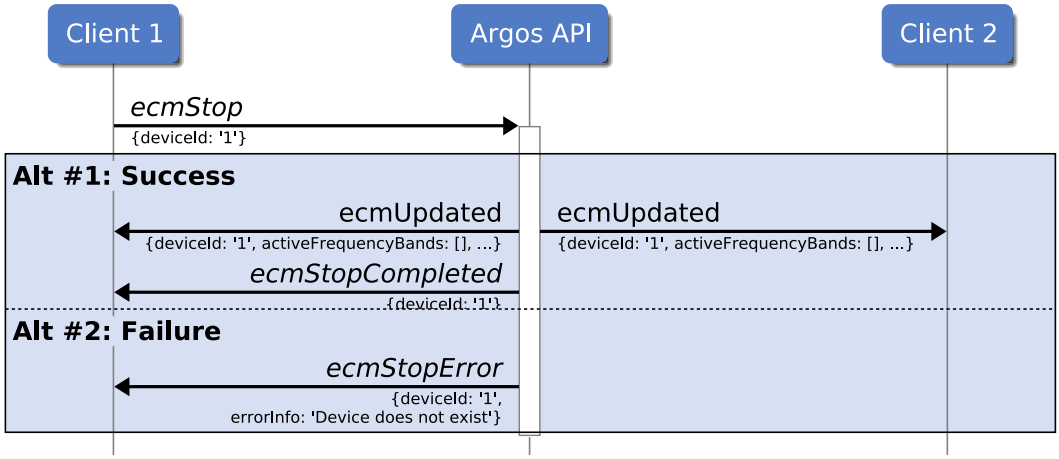

Figure 12.3: Example of ecmStop message communication between the client and the ARGOS API.

To stop an ECM on a device the following message must be sent. The device will then stop the ECM. Only available in manual mode.

Request: ecmStop
It is possible to send a request to stop an effector jamming.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

Response: ecmStopCompleted
When a «ecmStop» request command is successfully registered it will respond with a «ecmStopCompleted» response and if an error occurs a «ecmStopError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/DeviceDeviceIdError.json`

12.3 ecmGetState

Pattern: Request/Response

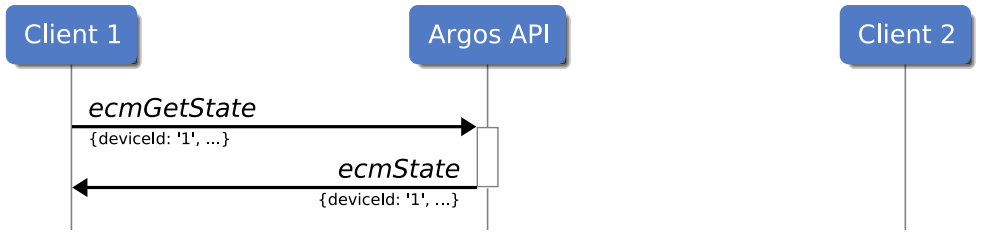

Figure 12.4: Example of ecmGetState message communication between the client and the ARGOS API.

To get the current state of an ECM device the following message must be sent. The device will then return the current state of the ECM device.

Request: ecmGetState
It is possible to send a request to get the current state of an effector.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json`

Response: ecmState
When a «ecmGetState»» request command is successfully registered it will respond with a «ecmState» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/EcmState.json`

12.4 ecmUpdated

Pattern: Push

When an ecm device is updated by e.g. an «ecmStart» an «ecmUpdated» message is pushed to all connected clients.

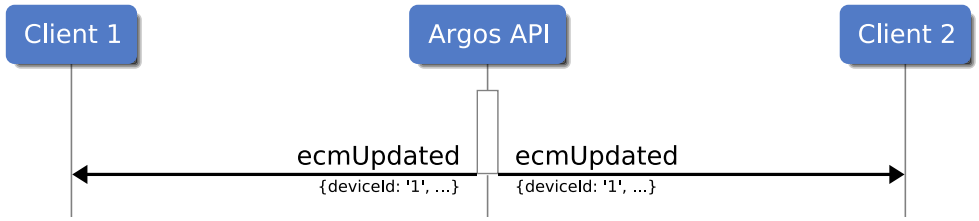

Figure 12.5: Example of ecmUpdated message communication from ARGOS API to a connected client.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/EcmState.json`

12.5 ecmGetHistory

Pattern: Request/Response

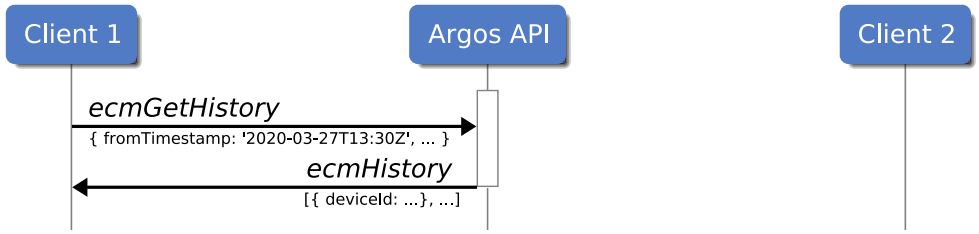

Figure 12.6: Example of ecmGetHistory message communication between the client and the ARGOS API.

Request: ecmGetHistory
Historical ECM events are logged in ARGOS and can be fetched using «ecmGetHistory».

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/TimeInterval.json`

Response: ecmHistory
When a «ecmGetHistory»» request command is successfully registered it will respond with a «ecmHistory» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/EcmHistory.json`

12.6 Automatic Jamming

The ECM service can handle requests to a virtual device "auto", which will start/stop automatic ECM on all devices according to the configuration. The "auto" device can be used with the commands ecmStart, ecmStop, ecmUpdated, and ecmGetState.

12.6.1 ecmConfigure

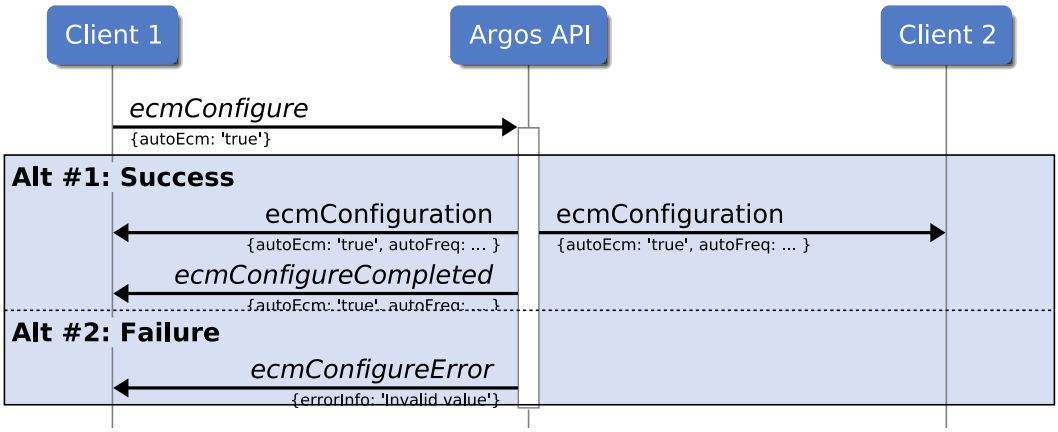

Figure 12.7: Example of ecmConfig message communication between the client and the ARGOS API.

The functionality of the "auto" device is configurable.

Request: ecmConfigure
Set or query the configuration (see table below).

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/EcmConfig.json`

Response: ecmConfigureCompleted
When an «ecmConfigure» request command is successfully handled it will respond with an «ecmConfigureCompleted» response containing all configuration parameters, or if an error occurs an «ecmConfigureError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/EcmConfig.json` or
`schema/messages/Error.json`

The «ecmConfigure» message must contain one or more of the following properties. The auto* properties are only used with the virtual device "auto".

Name	Type	Default	Description
autoStart	"off" "all" [zones]	"off"	Allow ARGOS to autonomously activate ECM. Activation can be triggered always or when the threat is identified to be in one of the listed alarm zone IDs.
autoFreq	"off" "noGnss" "allowGnss"	"off"	Let ARGOS continuously determine the best frequencies to jam. If set to "off", the configured frequency bands for all individual devices will be used. If no RF information is available (e.g. radar), 2G4 and 5G8 will be jammed. If set to "allowGnss" (and GNSS is supported by the ECM device) GNSS will always have first priority to be jammed. If set to "noGnss" GNSS jamming is never activated. Enabling GNSS jamming will for some ECM device reduces the number of other frequency bands, that can be jammed.
auto-Timeout	int	120	Used after ecmStart(deviceId="auto"): With autoStart=false jamming will stop after auto-Timeout seconds with no drone detection, and requires a new ecmStart to start again. With autoStart=true jamming will stop after auto-Timeout seconds with no drone detection. If a drone is later detected ECM will be started again.

The configuration will be stored and loaded in case of ARGOS restart.

«ecmConfiguration» always contains all configuration parameters

12.6.2 ecmConfiguration

Pattern: Push

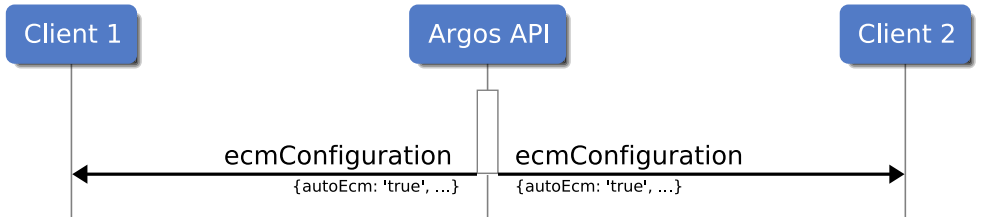

Figure 12.8: Example of ecmConfiguration message communication from ARGOS API to a connected client.

When the ECM configuration is updated by an «ecmConfigure» an «ecmConfiguration» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/EcmConfig.json`

12.6.3 ecmGetConfiguration

Pattern: Request/Response

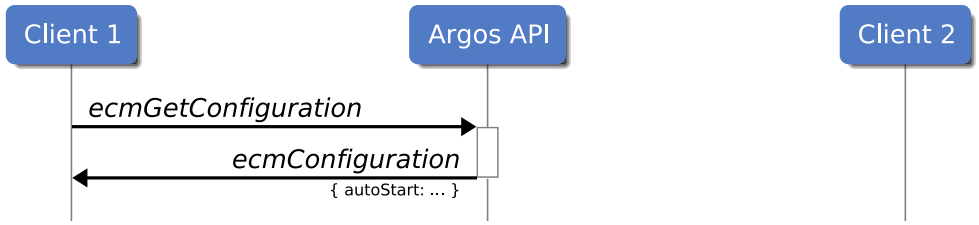

Figure 12.9: Example of ecmGetConfiguration message communication between the client and the ARGOS API.

To get the current ECM configuration the following message must be sent. The device will then return the current ECM configuration.

Request: ecmGetConfiguration
It is possible to send a request to get the current ECM configuration.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: ecmConfiguration
When an «ecmGetConfiguration»» request command is successfully registered it will respond with an «ecmConfiguration» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/EcmConfig.json`

12.6.4 Semi Automatic Operation

In a semi automatic use case, the operator is notified of an alert, and actively has to enable jamming. In the following example ARGOS selects frequencies because ecmConfig.autoFreq is set.

Current implementation always uses all ECM devices. With ecmConfig.autoFreq = "off", the operation is (currently) equivalent to sending ecmStart to all individual ECM devices. In the future it is possible that ARGOS will use a subset ECM devices when using the "auto" device.

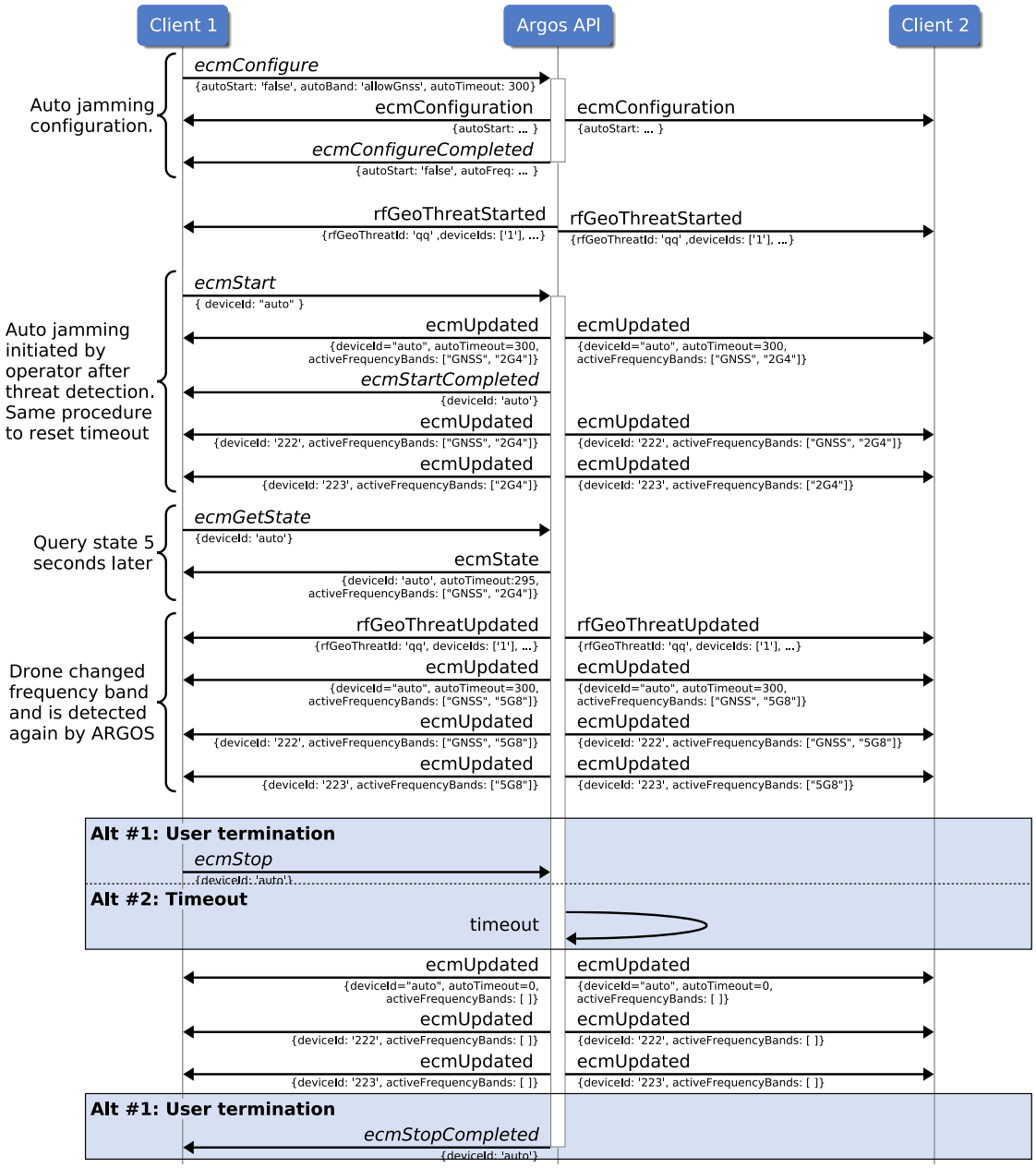

Figure 12.10: Example of message flow using the "auto" device to control jamming.

Timeout is configurable (ecmConfig.autoTimeout). In these examples 2 minutes is assumed. Jamming will continue for the specified time. The time may be extended by user interaction, or by the system detecting a drone.

Sending ecmStart(deviceId="auto") will only start jamming if drones are detected by the system. To force jamming send ecmStart to the individual ECM devices.

Example: No Drone Detection During Jamming

1. Drone detected.
2. Operator enables jamming.
3. Timeout is set to 2 minutes.

4. Due to jamming, the system cannot detect the drone.
5. After the timeout, the system stops the jamming, but the drone is still present.
6. System is now back to initial state.
7. Drone detected. Goto 2. (*Operator must manually re-enable jamming*).

Example: Drone Detection During Jamming

1. Drone detected.
2. Operator enables jamming.
3. Timeout is set to 2 minutes.
4. The system can detect the drone on a non-jammed frequency, or using radar.
   While detection goto 3.
5. After the timeout, the system stops the jamming.
6. System is now back to initial state.

Example: Operator Can See the Drone

1. Drone detected.
2. Operator enables jamming.
3. Timeout is set to 2 minutes.
4. Due to jamming, the system cannot detect the drone.
5. The operator visually follows the Drone.
   While visual contact operator regularly clicks 'extend'. Goto 3.
6. After the timeout, the system stops the jamming.
7. System is now back to initial state.

12.6.5 Fully Automatic Operation

ARGOS autonomously enables jamming when a threat is detected. ARGOS selects frequencies if ecmConfig.autoFreq is enabled.

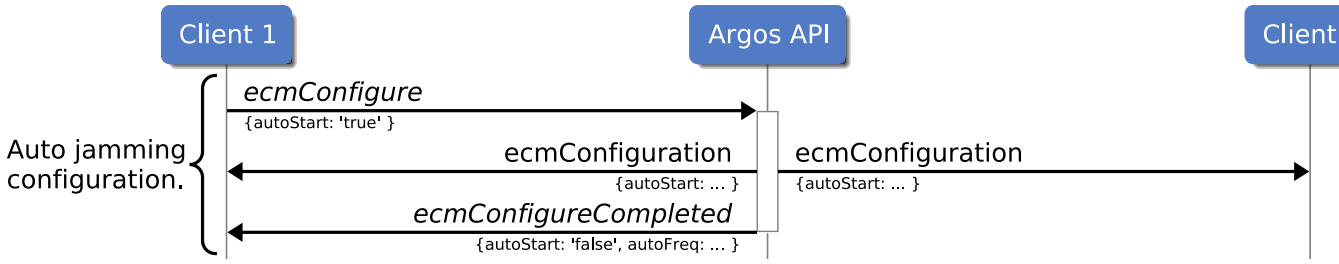

Figure 12.11: Configuration for fully automatic jamming.

The message flow is identical to the semi automatic flow in 12.10, with the following exceptions:

- The ecmConfigure settings.
- No ecmStart and ecmStartCompleted messages.

«ecmStop» is normally not needed when ecmConfig.autoStart is enabled. It may be used, but if the threat is still present jamming will be restarted. To completely stop jamming ecmConfig.autoStart must be disabled.

13. System-state

Several messages are used to inform the API client of the state of the system.

13.1 getSystemState

Pattern: Request/Response

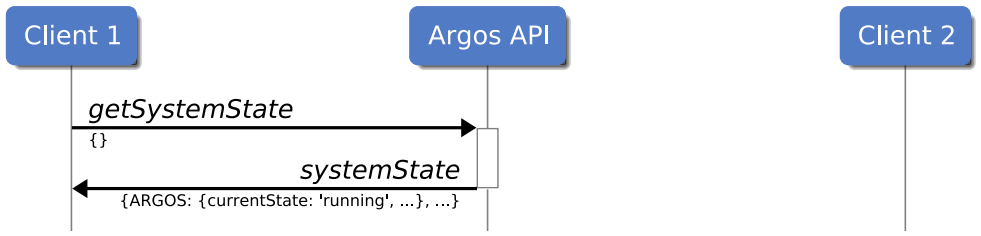

Figure 13.1: Example of getSystemState message communication between the client and the ARGOS API.

It is possible to fetch state of the backend system on the server by sending the event «getSystemState». ARGOS will respond with the «systemState» message, which is an overview of the system. Contained in the message are fields describing the state of each sub-component of ARGOS.

Request: getSystemState
It is possible to send a request to get the current state of the backend system on the server.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: systemState
When a «getSystemState» request command is successfully registered it will respond with a «systemState» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/SystemState.json`

13.2 getTime

Pattern: Request/Response

It is possible to fetch the time of the server by sending the event «getTime». ARGOS will respond with the time formatted in ISO8601-format.

Request: getTime

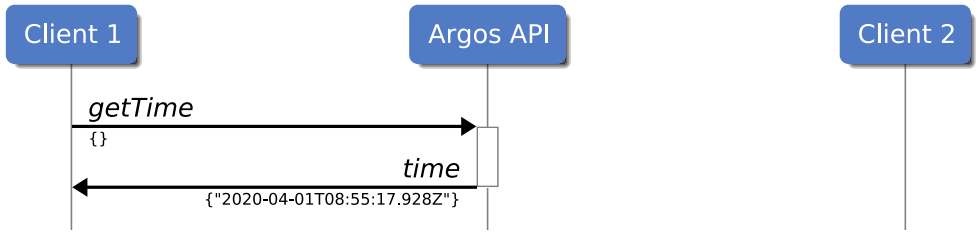

Figure 13.2: Example of getTime message communication between the client and the ARGOS API.

It is possible to send a request to get the current server time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: time
When a «getTime» request command is successfully registered it will respond with a «time» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Time.json`

13.3 systemLog

Pattern: Push

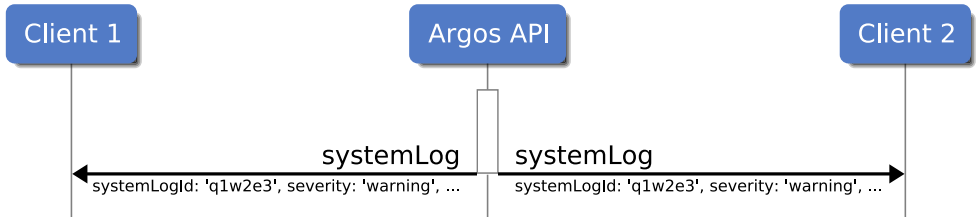

Figure 13.3: Example of systemLog message communication from ARGOS API to a connected client when a new alarm zone has been created.

System Log messages will be pushed to clients to notify of errors and warnings detected in the system.

For example it may be that one sensor is picking up RF interference or contact to equipment has been lost etc.

When a system event is created a «systemLog» is pushed where createdTimeStamp and updatedTimeStamp are identical and resolvedTimeStamp is not present. If the event is updated e.g. a new severity or description, then createdTimeStamp will keep the original value while updatedTimeStamp is updated.

When a system event is resolved the «systemLog» is pushed including the resolvedTimeStamp.

13.4 systemLogGetList

Pattern: Request/Response

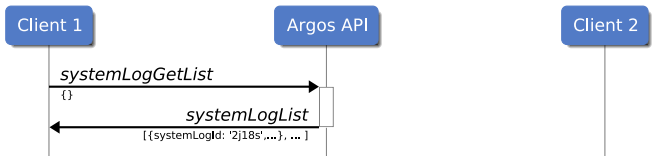

Figure 13.4: Example of systemLogGetList message communication between the client and the ARGOS API.

«systemLogList» contains an array of all active error- and warning-events in the system.

When using the «systemLogGetList» command a response with «systemLogList» will be sent back as response.

13.5 systemLogList

Pattern: Push

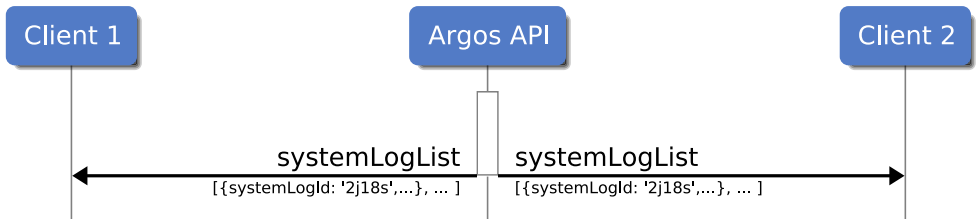

Figure 13.5: Example of systemLogList message communication from ARGOS API to a connected client when a new alarm zone has been created.

Upon service start «systemLogList» will be pushed to all connected clients.

During ARGOS startup or if an unforeseen error causes a restart of the internal service an «systemLogList» message is pushed to all connected clients. This list can also be requested via «systemLogGetList» described in section 13.4.

14. PTZ

The PTZ service provides generic control of pan-tilt-zoom devices (including focus control), such as rotators for jammers or video cameras.

14.1 ptzGetDeviceList

Pattern: Request/Response

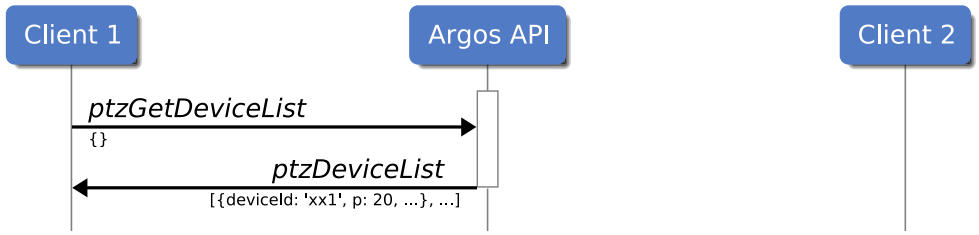

Figure 14.1: Example of the ptzGetDeviceList message communication between the client and the ARGOS API.

Request: ptzGetDeviceList
Request a list of all connected PTZ devices and their capabilities. Entries in the list has the same format as «ptzDeviceInfo».

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: ptzDeviceList
When a «ptzGetDeviceList» request command is successfully registered a «ptzDeviceList» response will be returned.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzDeviceList.json`

Changes to the list will be signalled with «ptzDeviceInfo».

14.2 ptzDeviceInfo

Pattern: Push

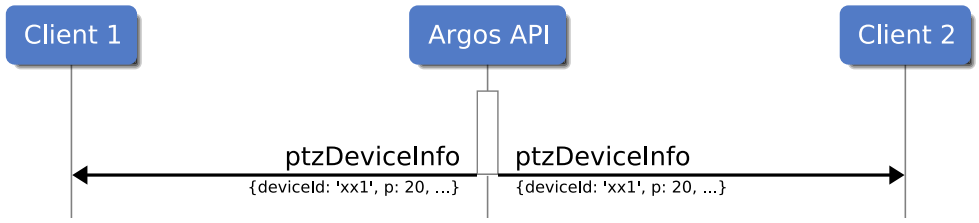

Figure 14.2: Example of ptzDeviceInfo message communication from ARGOS API to a connected client.

When a PTZ device comes online a «ptzDeviceInfo» message is pushed to all connected clients. From this message current setting of all controllable dimensions of the PTZ device can be seen, and thus also which dimensions can be controlled.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzDeviceInfo.json`

14.3 ptzDeviceUpdate

Pattern: Push

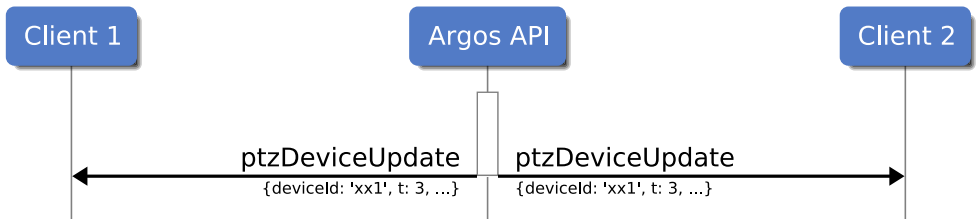

Figure 14.3: Example of ptzDeviceUpdate message communication from ARGOS API to a connected client.

When a PTZ device is changed, e.g. moved using a joystick, a «ptzDeviceUpdate» is used to inform of the changed values. The format is the same as «ptzDeviceInfo», but only changed values are included.

Also see the MSCs for «ptzMoveAbs» in fig. 14.4 on the next page and for «ptzSetVelocity» in fig. 14.5 on page 80.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzDeviceInfo.json`

14.4 ptzMoveAbs

Pattern: Request/Response

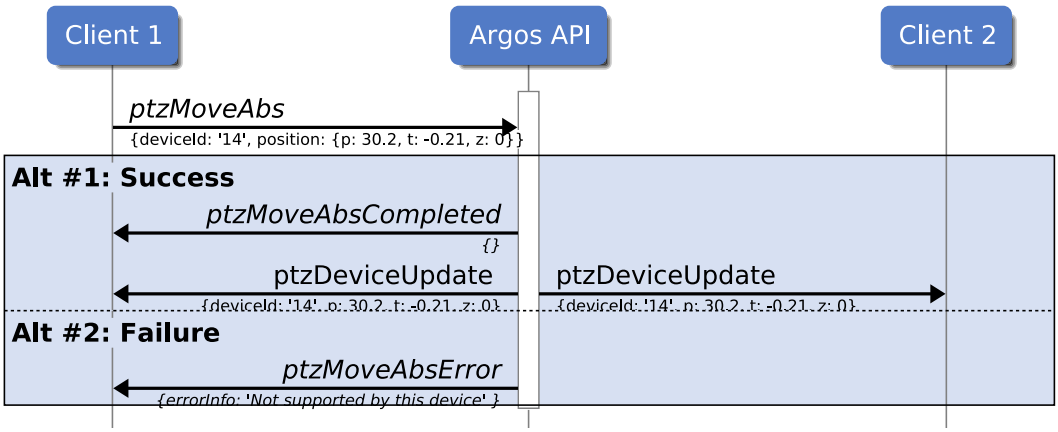

Figure 14.4: Example of the ptzMoveAbs message communication between the client and the ARGOS API

Request: ptzMoveAbs
Request ARGOS to move a PTZ device to a specific position as sepcified in «ptzMoveAbs» request.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzMoveAbs.json`

Response: ptzMoveAbsCompleted
When a «ptzMoveAbs» request command is successfully registered a «ptzMoveAbsCompleted» response will be returned. If the request was unsuccessful a «ptzMoveAbsError» will be returned.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/Error.json`

14.5 ptzSetVelocity

Pattern: Request/Response

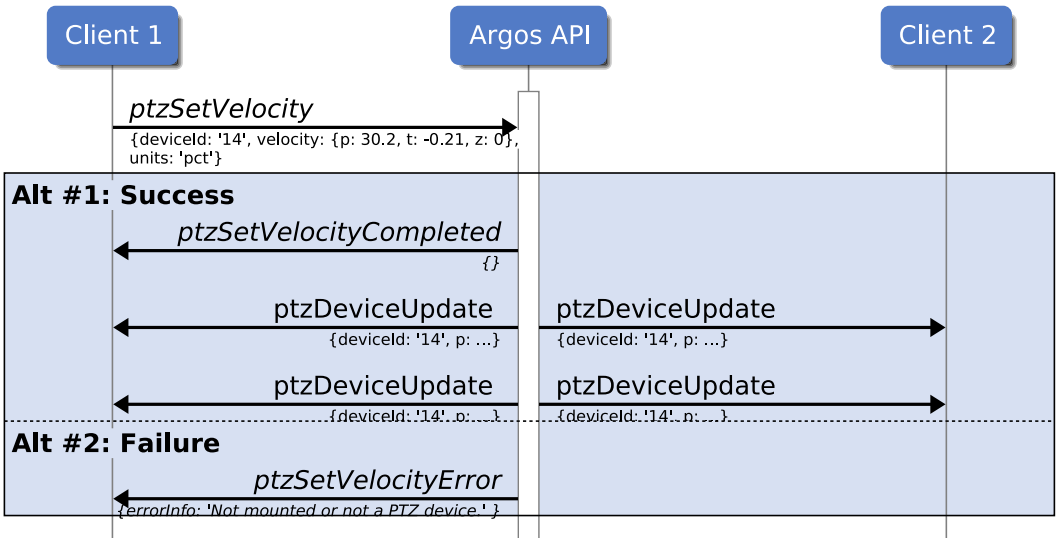

Figure 14.5: Example of the ptzSetVelocity message communication between the client and the ARGOS API.

Request: ptzSetVelocity
Request ARGOS to move a PTZ device at a velocity as specified in «ptzSetVelocity» request.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzSetVelocity.json`

'units' define the units of 'velocity' and can be either 'deg/s' or 'pct'. Default value of 'units' is 'pct'.

Note some devices may not support all 'units' options. If a 'units' is not supported «ptzSetVelocityError» will be returned.

Response: ptzSetVelocityCompleted
When a «ptzSetVelocity» request command is successfully registered a «ptzSetVelocityCompleted» response will be returned. If the request was unsuccessful a «ptzSetVelocityError» will be returned.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/Error.json`

14.6 ptzDeviceSet

Pattern: Request/Response

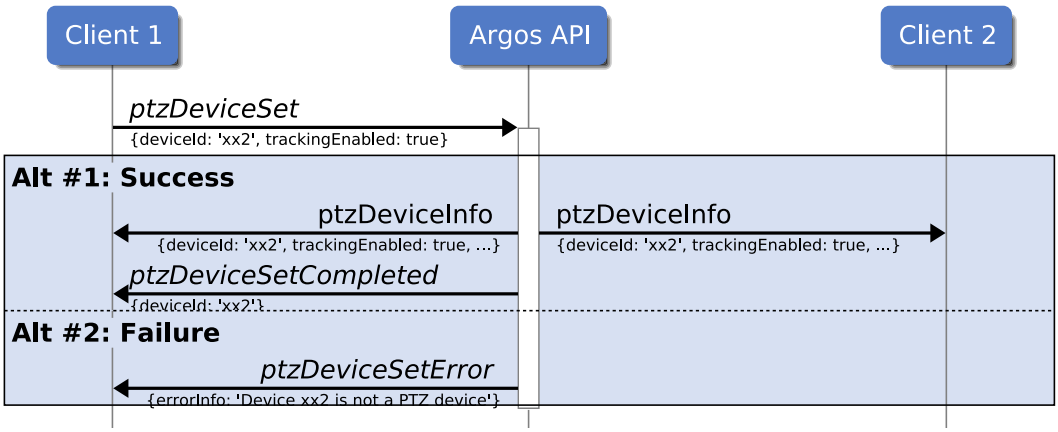

Figure 14.6: Example of the ptzDeviceSet message communication between the client and the ARGOS API.

Request: ptzDeviceSet
Set various features of PTZ devices.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzDeviceSet.json`

Response: ptzDeviceSetCompleted
When a «ptzDeviceSet» request command is successfully registered a «ptzDeviceSet-Complete» response will be returned. If the request was unsuccessful a «ptzDevice-SetError» will be returned.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceId.json` or
`schema/messages/Error.json`

14.7 PTZ Following

PTZ The service can be configured to automatically use a PTZ device to follow a threat.

The overall flow of automatic following threats is shown in fig. 14.7.

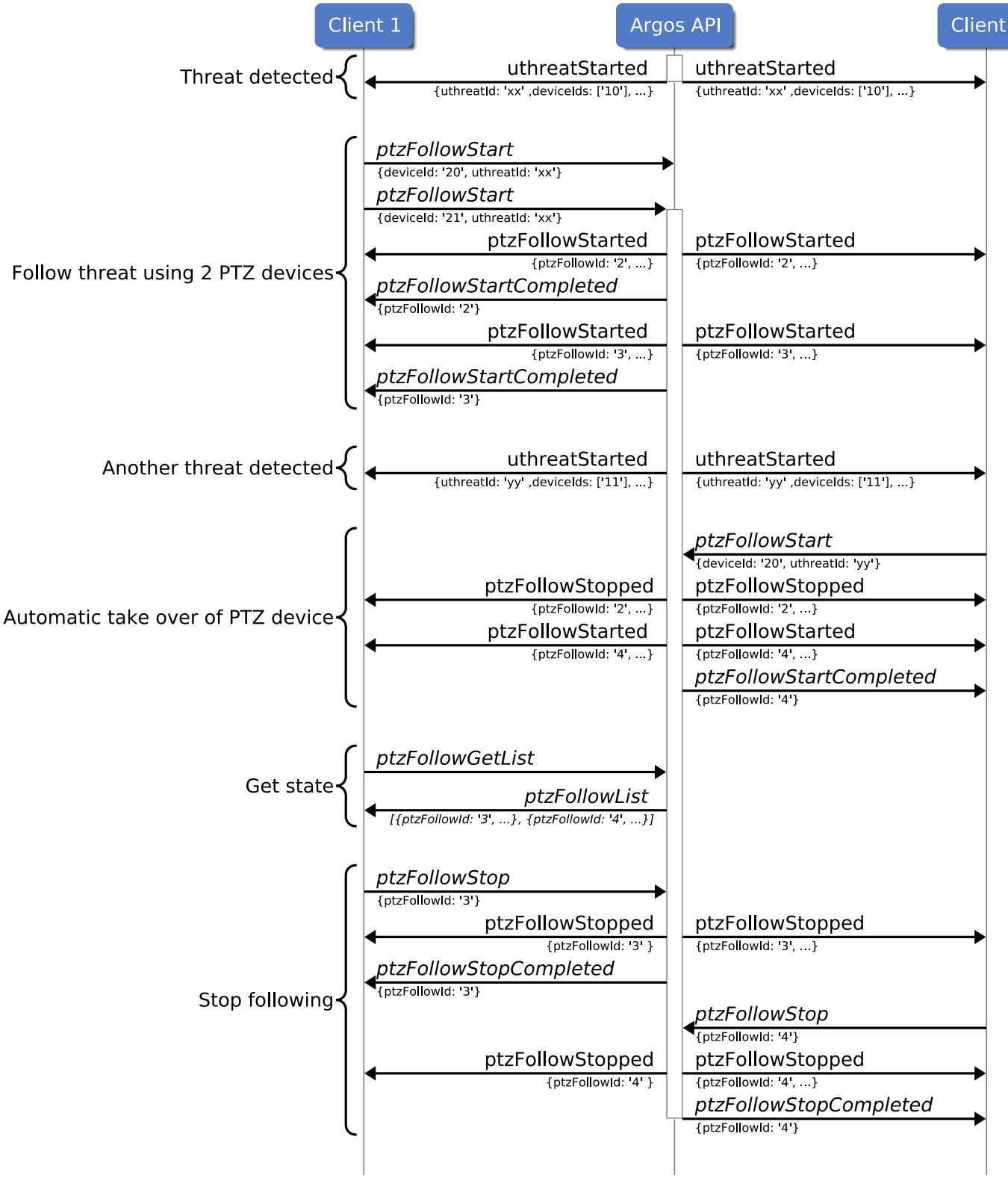

Figure 14.7: Example PTZ control communication between the client and the ARGOS API.

14.8 ptzFollowStart

Pattern: Request/Response

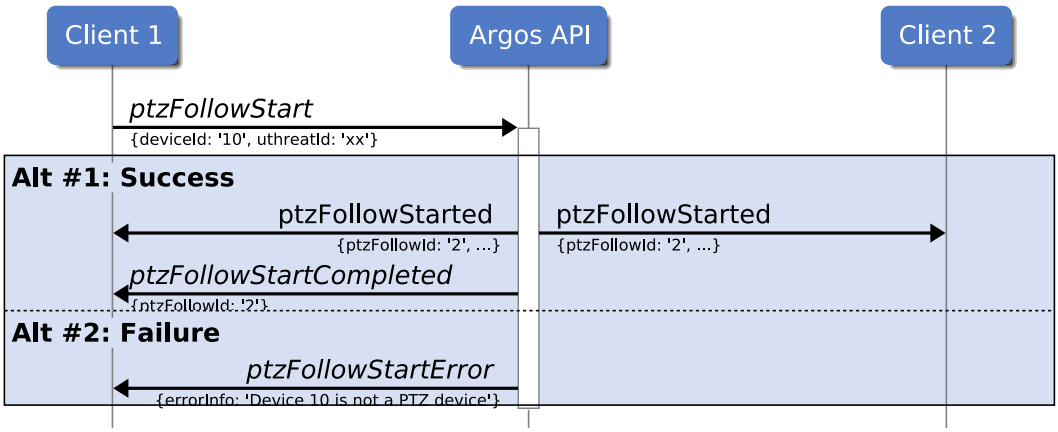

Figure 14.8: Example of the PTZ ptzFollowStart message communication between the client and the ARGOS API.

Request ARGOS to use a PTZ device for following a threat with the «ptzFollowStart» request. When using the «ptzFollowStart» command a response with «ptzFollowStart-Completed» will be sent back to show that the follow service has been activated. The operation causes a push message with a «ptzFollowStarted» to be sent with information about the follow configuration. In case an error occurs during the «ptzFollowStart» a «ptzFollowStartError» message will be sent as response.

Request: ptzFollowStart
The «ptzFollowStart» request will activate the PTZ follow service.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollowStart.json`

Note: One and only one of «ptzFollowStart»'s properties followUthreatId / followDeviceId of must be set.

When the «ptzFollowStart» request has been processed correctly a «ptzFollowStarted» message will be pushed with information about the new PTZ follow instance.

Response: ptzFollowStartCompleted
When a «ptzFollowStart» request command is successfully registered it will respond with a «ptzFollowStartCompleted» response and if an error occurs a «ptzFollowStartError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollowStartCompleted.json` or
`schema/messages/Error.json`

14.9 ptzFollowStarted

Pattern: Push

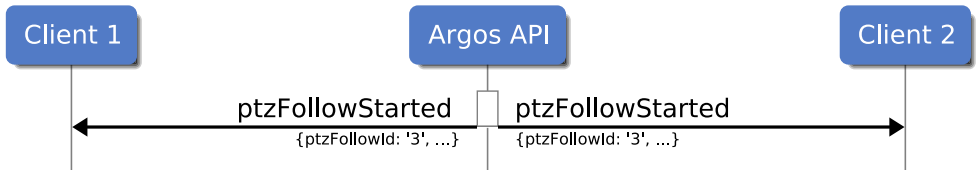

Figure 14.9: Example of the PTZ ptzFollowStarted message communication between the client and the ARGOS API.

When a new PTZ follow instance is created a «ptzFollowStarted» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollow.json`

14.10 ptzFollowStop

Pattern: Request/Response

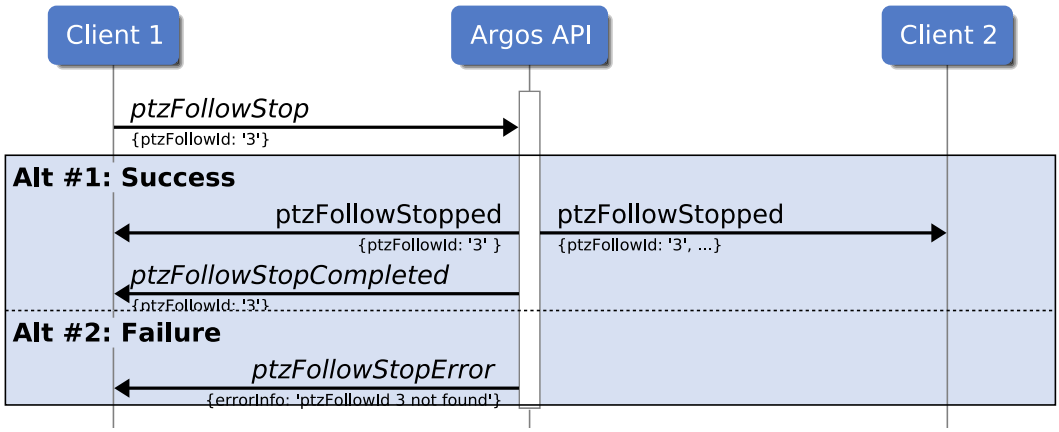

Figure 14.10: Example of the ptzFollowStop message communication between the client and the ARGOS API.

Request: ptzFollowStop
Request to end automatic following.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollowId.json`

Response: ptzFollowStopCompleted
When a «ptzFollowStop» request command is successfully handled it will respond with a «ptzFollowStopCompleted» response and if an error occurs a «ptzFollowStopError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollowId.json` or
`schema/messages/Error.json`

14.11 ptzFollowStopped

Pattern: Push

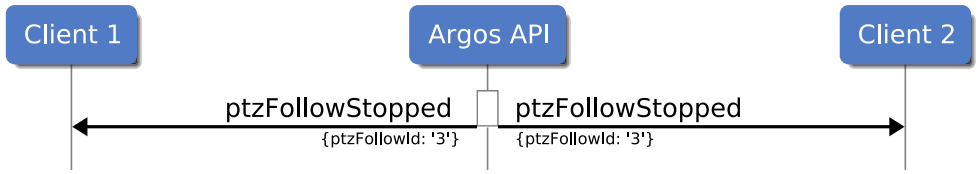

Figure 14.11: Example of the ptzFollowStop message communication between the client and the ARGOS API.

When a PTZ follow instance has been stopped «ptzFollowStopped» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollow.json`

14.12 ptzFollowGetList

Pattern: Request/Response

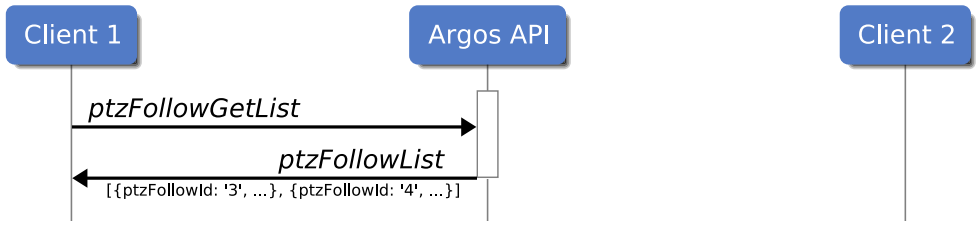

Figure 14.12: Example of the ptzFollowGetList message communication between the client and the ARGOS API.

Request: ptzFollowGetList
A list of all active PTZ follow instances can be fetched using «ptzFollowGetList».

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: ptzFollowList
The list of all active instances.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollowList.json`

14.13 ptzFollowList

Pattern: Push

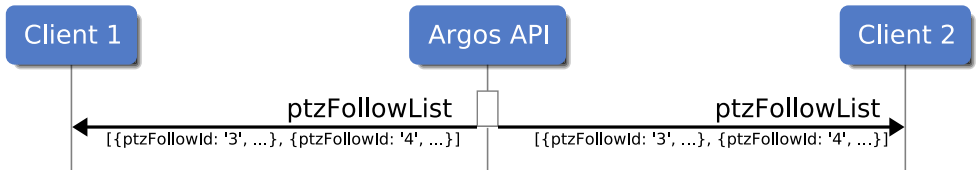

Figure 14.13: Example of the ptzFollowList message communication between the client and the ARGOS API.

During ARGOS startup or if an unforeseen error causes a restart of the internal PTZ service a «ptzFollowList» message is pushed to all connected clients. The list describes all the currently active PTZ follow instances. This list can also be requested via «ptzFollowGetList» described in section 14.12 on the preceding page.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/PtzFollowList.json`

15. Firmware Update

The Firmware Update API is **deprecated**. It will be kept for foreseeable future, but it is only needed to update device without MDIF support, i.e. RF sensors with SW version prior to 2.14, and RF effector with SW version prior to version 1.5.

For newer devices with MDIF support use the MyDefence Device Manager for device firmware updates.

The MyDefence devices connected to and controlled (mounted) by an ARGOS instance can be firmware updated (FWU). Firmware images for individual device types (RF sensors and RF effectors) can be uploaded to ARGOS. ARGOS stores these persistently continuously checks if devices in the connected state are eligible for FWU. I such is found the clients connected to ARGOS are notified. A client can then initiate the actual FWU process. The overall flow of managing FWU images is shown in fig. 15.1.

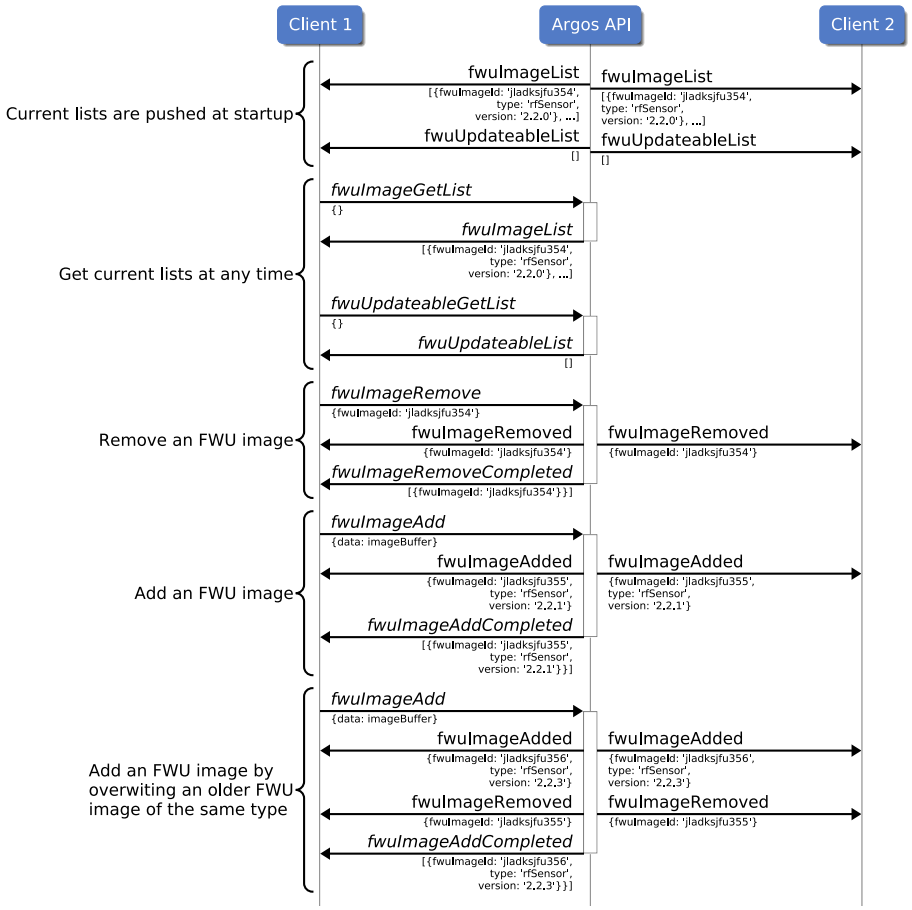

Figure 15.1: Example session showing how FWU images can be managed.

As previously stated ARGOS monitors FWU images and connected devices. When a device is in the list of updateable devices, the client can initiate the FWU process. The FWU process flow is shown in fig. 15.2 on the following page.

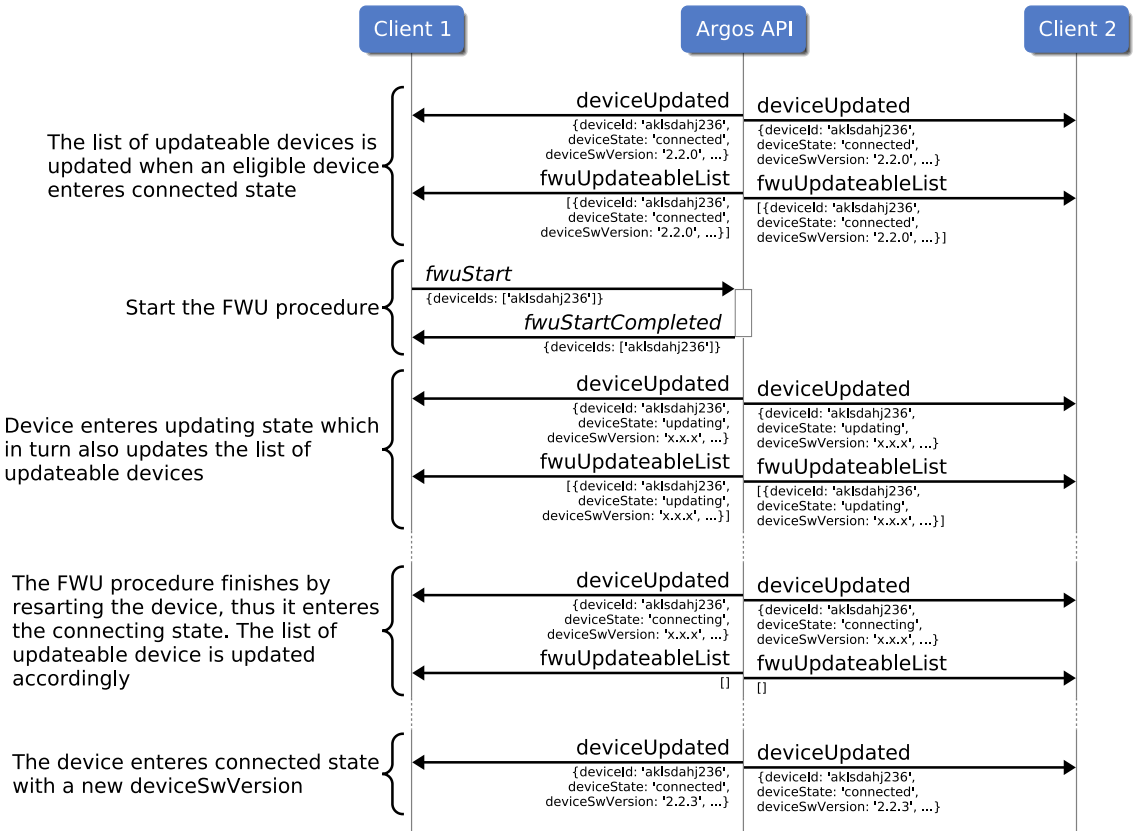

Figure 15.2: Example session showing how the FWU process is initiated.

15.1 fwuImageAdd

Pattern: Request/Response

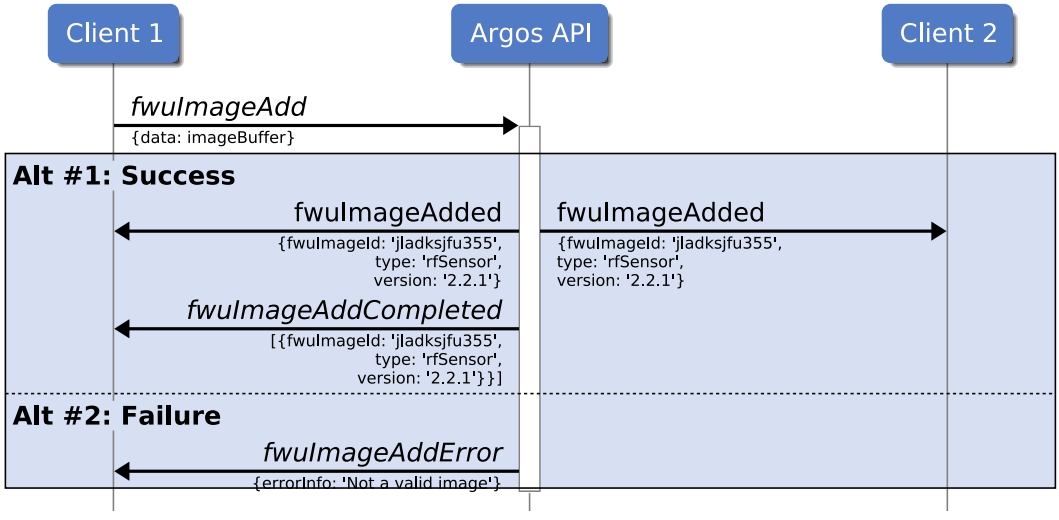

Figure 15.3: Example of fwuImageAdd message communication between the client and the ARGOS API.

When an FWU image is added an «fwuImageAdd» command must be sent and a response with «fwuImageAddCompleted» will be send back to show that the image has been added. The operation causes a push message with an «fwuImageAdded»

to be sent with information about the image that has been added. In case an error occurs during the «fwuImageAdd» an «fwuImageAddError» message will be sent as response.

Request: fwuImageAdd
It is possible to send a request to upload a supported FWU image and have it added to the list of available FWU images.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImageAdd.json`

When the «fwuImageAdd» request has been processed an «fwuImageAdded» message will be pushed with information about the added image.

Response: fwuImageAddCompleted
When an «fwuImageAdd» request command is successfully registered it will respond with an «fwuImageAddCompleted» response and if an error occurs an «fwuImageAddError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImage.json` or
`schema/messages/Error.json`

15.2 fwuImageRemove

Pattern: Request/Response

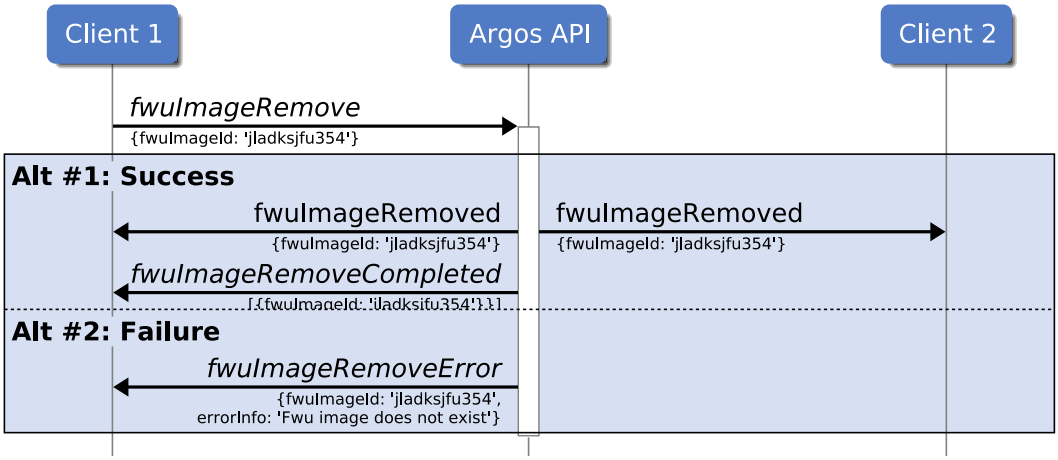

Figure 15.4: Example of fwuImageRemove message communication between the client and the ARGOS API.

When an FWU image needs to be removed an «fwuImageRemoved» command must be sent and a response with «fwuImageRemoveCompleted» will be send back to show that the image has been removed. The operation causes a push message with an «fwuImageRemoved» to be sent with information about the image that has been removed. In case an error occurs during the «fwuImageRemove» an «fwuImageRe-

moveError» message will be sent as response.

Request: fwuImageRemove
It is possible to send a request to remove a previous added device. When an «fwuImageRemove» request is processed all data related to the FWU image is deleted and can not be recovered afterwards.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImageId.json`

When the «fwuImageRemove» request has been processed an «fwuImageRemoved» message will be pushed with information about the image.

Response: fwuImageRemoveCompleted
When an «fwuImageRemove» request command is successfully registered it will respond with an «fwuImageRemoveCompleted» response and if an error occurs an «fwuImageRemoveError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImageId.json` or
`schema/messages/Error.json`

15.3 fwuImageGetList

Pattern: Request/Response

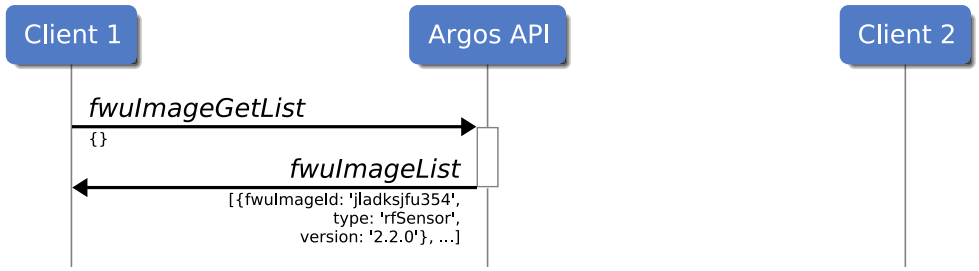

Figure 15.5: Example of fwuImageGetList message communication between the client and the ARGOS API.

A list of the currently stored FWU images can be requested via an «fwuImageGetList» command. ARGOS sends a response with «fwuImageList» back to the requesting client.

Request: fwuImageGetList
An «fwuImageGetList» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: fwuImageList

When an «fwuImageGetList» request command is successfully registered it will respond with an «fwuImageList» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImageList.json`

15.4 fwuUpdateableGetList

Pattern: Request/Response

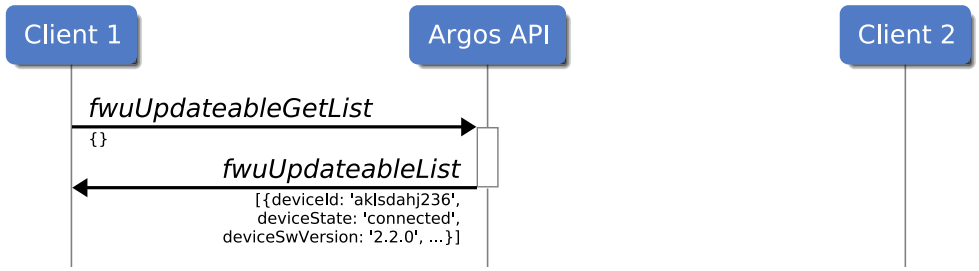

Figure 15.6: Example of fwuUpdateableGetList message communication between the client and the ARGOS API.

A list of the currently updateable devices can be requested via an «fwuUpdateableGetList» command. ARGOS sends a response with «fwuUpdateableList» back to the requesting client. The list describes all the currently updateable devices. A device is eligible for update if a stored FWU image supports the device type and the image version is greater than the version running in the device.

Request: fwuUpdateableGetList
An «fwuUpdateableGetList» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: fwuUpdateableList
When an «fwuUpdateableGetList» request command is successfully registered it will respond with an «fwuUpdateableList» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuUpdateableList.json`

15.5 fwuImageAdded

Pattern: Push

When a new FWU image is added an «fwuImageAdded» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:

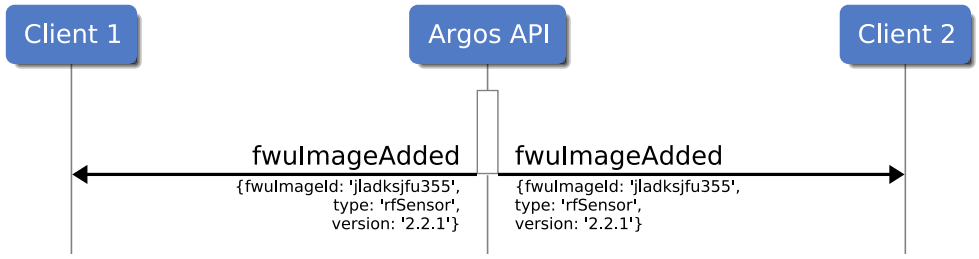

Figure 15.7: Example of fwuImageAdded message communication from ARGOS API to a connected client when a new FWU image has been added.

`schema/messages/FwuImage.json`

15.6 fwuImageRemoved

Pattern: Push

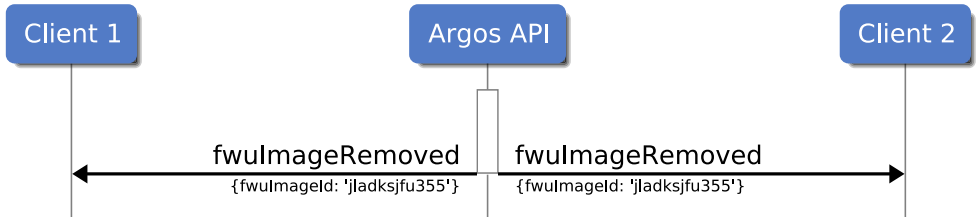

Figure 15.8: Example of fwuImageRemoved message communication from ARGOS API to a connected client.

When an FWU image is removed by e.g. an «fwuImageRemove» an «fwuImageRemoved» message is pushed to all connected clients.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImageId.json`

15.7 fwuImageList

Pattern: Push

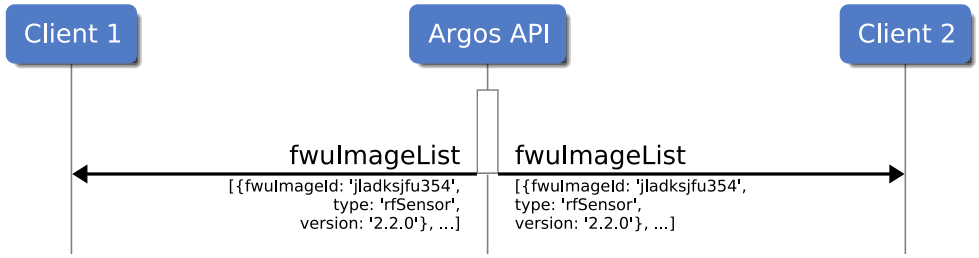

Figure 15.9: Example of fwuImageList message communication from ARGOS API to a connected client.

During ARGOS startup or if an unforeseen error causes a restart of the internal FWU

service an «fwuImageList» message is pushed to all connected clients. The list describes all the currently stored FWU images. This list can also be requested via «fwuImageGetList» described in section 15.3 on page 90.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImageList.json`

15.8 fwuUpdateableList

Pattern: Push

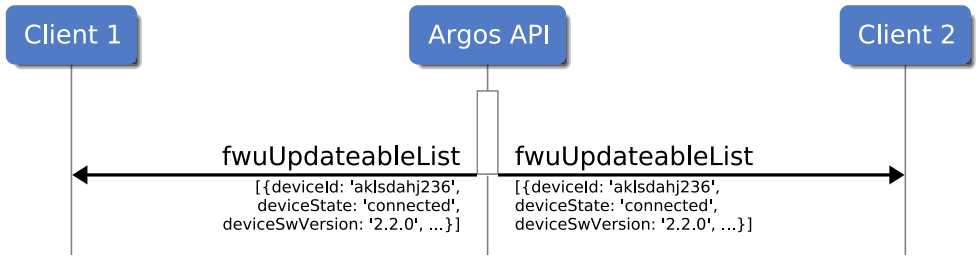

Figure 15.10: Example of fwuUpdateableList message communication from ARGOS API to a connected client.

During ARGOS startup or if an unforeseen error causes a restart of the internal FWU service an «fwuUpdateableList» message is pushed to all connected clients. The list describes all the currently updateable devices. A device is eligible for update if a stored FWU image supports the device type and the image version is greater than the version running in the device. This list can also be requested via «fwuUpdateableGetList» described in section 15.4 on page 91.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FwuImageList.json`

15.9 fwuStart

Pattern: Request/Response

The FWU process can be started via an «fwuStart» command. ARGOS sends a response with «fwuStartCompleted» back to the requesting client to show that the the process has been started. In case an error occurs during the «fwuStart» an «fwuStartError» message will be sent as response. During the FWU process the state of the updating devices will change. «deviceUpdated» messages will be pushed to all connected clients. The device states will cycle through "updating", "connecting" to "connected".

Request: fwuStart
It is possible to send a request to start the FWU process.

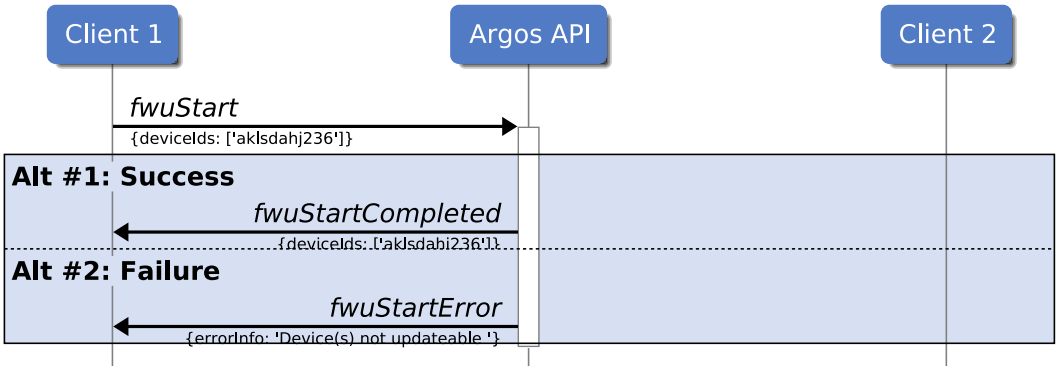

Figure 15.11: Example of fwuStart message communication between the client and the ARGOS API.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceIds.json`

When the «fwuStart» request has been processed an «fwuStarted» message will sent back to the requesting client.

Response: fwuStartCompleted
When an «fwuStart» request command is successfully registered it will respond with an «fwuStartCompleted» response and if an error occurs an «fwuStartError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/DeviceIds.json` or
`schema/messages/Error.json`

16. rfGeoThreatSimulation

16.1 rfGeoThreatSimulationStart

Pattern: Request/Response

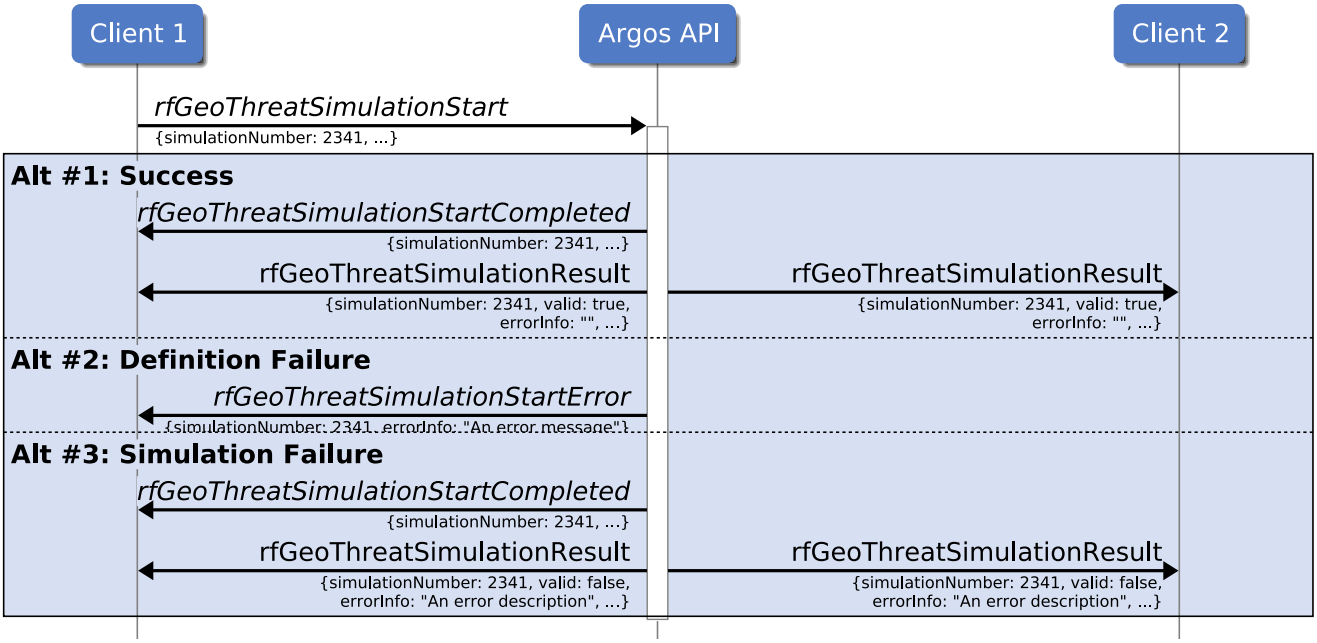

Figure 16.1: Example of rfGeoThreatSimulation message communication between the client and the ARGOS API.

A simulation of the coverage for a given setup can be requested via a «rfGeoThreatSimulationStart» command. ARGOS sends a response with «rfGeoThreatSimulationStartCompleted» back to the requesting client. Once the calculations for the simulation has completed a «rfGeoThreatSimulationResult» message is pushed to all clients. In case an error occurs during the «rfGeoThreatSimulationStart» a «rfGeoThreatSimulationStartError» message will be sent as response. In case an error occurs during the calculation for a simulation a «rfGeoThreatSimulationResult» message with the property "valid" set to false is pushed to all clients.

Request: rfGeoThreatSimulationStart
A «rfGeoThreatSimulationStart» request can be sent at any time.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/RfGeoThreatSimulationStart.json`

Response: rfGeoThreatSimulationStartCompleted
When a «rfGeoThreatSimulationStart» request command is successfully registered it will respond with a «rfGeoThreatSimulationStartCompleted» response and if an error occurs a «rfGeoThreatSetMuteCompletedError» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/RfGeoThreatSimulationStartCompleted.json` or
`schema/messages/RfGeoThreatSimulationStartError.json`

16.2 rfGeoThreatSimulationResult

Pattern: Push

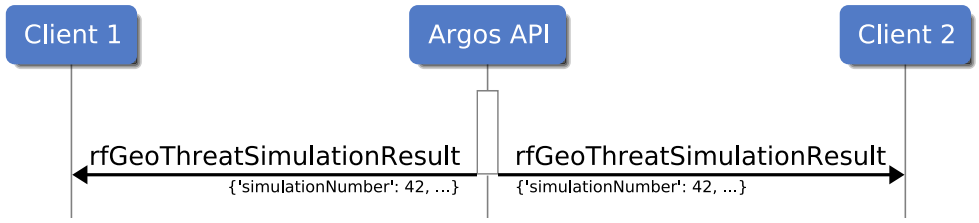

Figure 16.2: Example of rfGeoThreatSimulationResult message communication from ARGOS API to a connected client.

When a rfGeoThreatSimulation is done a «rfGeoThreatSimulationResult» message is sent.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/RfGeoThreatSimulationResult.json`

17. Mission Center

The mission center can be used by a user interface to know where to position a mission. In addition it is used internally in ARGOS, e.g. when evaluating threats and in auto jamming, to determine distance to threat. Mission center must therefore be set by the user of the API.

17.1 missionCenterSet

Pattern: Request/Response

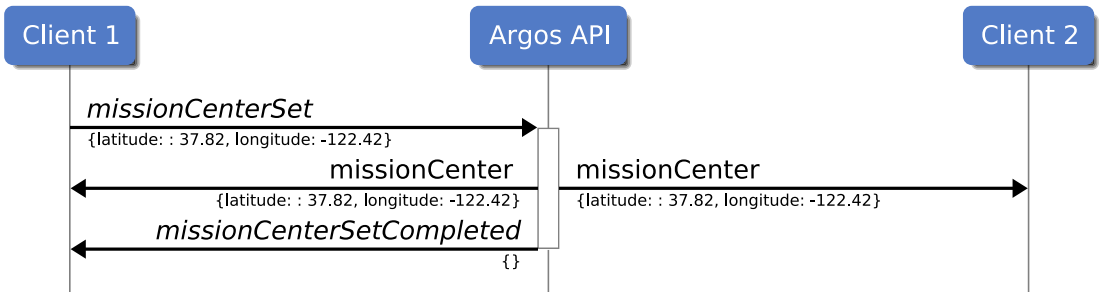

Figure 17.1: Example of missionCenterSet message communication between the client and the ARGOS API.

Request: missionCenterSet
Set the mission center. There are no range checks. All connected clients will be notified with a «missionCenter» push notification.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/missionCenter.json`

Response: missionCenterSetCompleted
Sent in response to «missionCenterSet».

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

17.2 missionCenterGet

Pattern: Request/Response

Request: missionCenterGet
Get the mission center.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

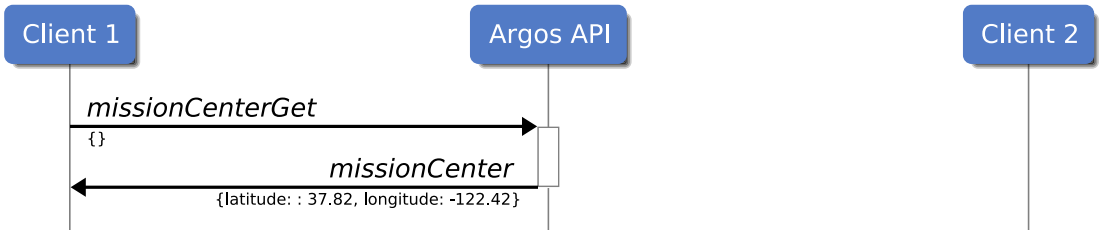

Figure 17.2: Example of missionCenterGet message communication between the client and the ARGOS API.

Response: missionCenter
Sent in response to «missionCenterGet».

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/missionCenter.json`

18. Video Management System

The ARGOS VMS (Video Management System) can be used to receive video from cameras. Some video streams will be relayed through ARGOS to enable multiple clients per camera. The video protocol used by ARGOS is WebRTC because it has very low latency. Other video streams are provided directly from the camera, and their protocol will differ from WebRTC.

WebRTC streams are started using the Session Description Protocol (SDP) (see section 18.3 on the next page), whereas other streams are started using the provided URI (see section 18.1).

Video devices may be auto discovered (if supported), or added using the normal device handling. When the device is mounted, the available streams are auto discovered. One camera device may provide multiple streams, e.g. a daylight and a thermal stream.

18.1 vmsStreamGetList

Pattern: Request/Response

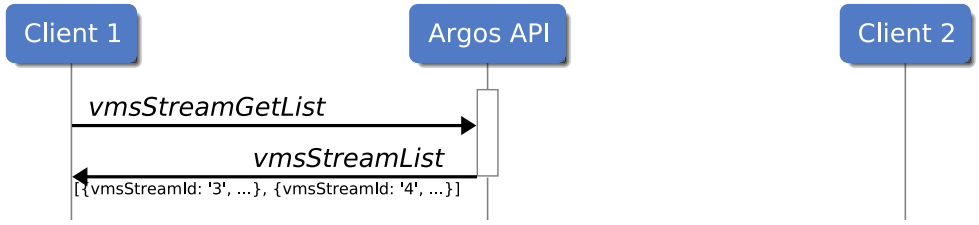

Figure 18.1: Example of the vmsStreamGetList message communication between the client and the ARGOS API.

Request: vmsStreamGetList
It is possible to send a request to get all video streams in ARGOS.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: vmsStreamList
When an «vmsStreamGetList» request command is successfully registered it will respond with an vmsStreamList response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/VmsStreamList.json`

18.2 vmsStreamList

Pattern: Push

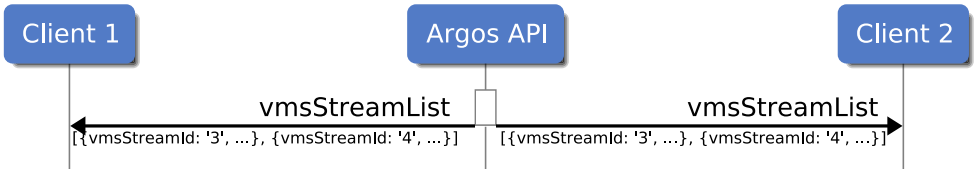

Figure 18.2: Example of the vmsStreamList message communication between the client and the ARGOS API.

During ARGOS startup and when a stream is added or removed, an «vmsStreamList» message is pushed to all connected clients, with the list of all known streams.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/VmsStreamList.json`

18.3 vmsSdpInit

Pattern: Request/Response

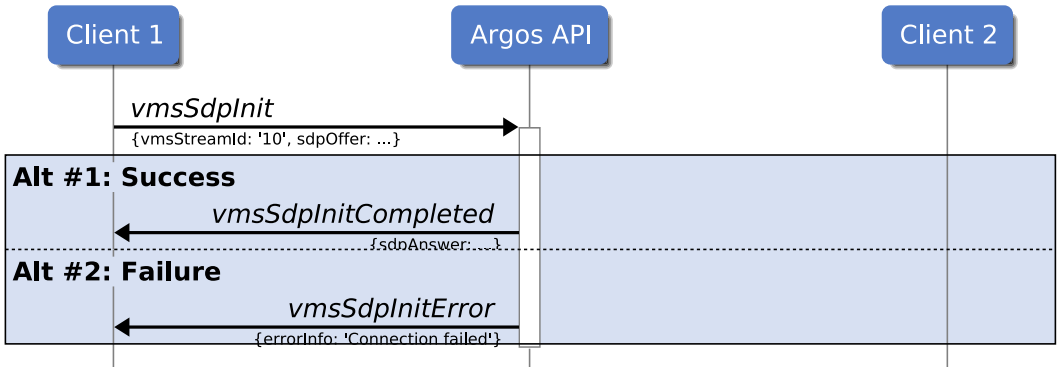

Figure 18.3: Example of the vmsSdpInit message communication between the client and the ARGOS API.

Request: vmsSdpInit
WebRTC uses the session description protocol (SDP) to establish a video link. The protoocol is described on webrtc.org.

The client must create an SDP offer using the RTCPeerConnection class, and send the offer to ARGOS using «vmsSdpInit». ARGOS will repond with the SDP reply in «vmsSdpInitCompleted» message. You may initialize the RTCPeerConnection with an empty config, because no ICE servers are needed on an internal network.

The following javascript code can be used in a client to get an SDP offer.

```
const peerConnection = new RTCPeerConnection()
```

```
peerConnection.addTransceiver('video', {
    'direction': 'recvonly'
})
const offer = await peerConnection.createOffer()
await peerConnection.setLocalDescription(offer)
const sdpOffer = peerConnection.localDescription.sdp
```

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/VmsSdpInit.json`

Response: vmsSdpInitCompleted
When an «vmsSdpInit» request command is successfully registered it will respond with an «vmsSdpInitCompleted» response and if an error occurs an vmsSdpInitError» response.

The «vmsSdpInitCompleted» response contains the SDP reply to be passed into WebRTC.

The following javascript code can be used in a client to get when receiving the SDP reply.

```
peerConnection.setRemoteDescription(
    new RTCSessionDescription({
        type: 'answer',
        sdp: sdpReply
}))
```

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/VmsSdpInitCompleted.json` or
`schema/messages/Error.json`

18.4 vmsBoundingBox

Pattern: Push

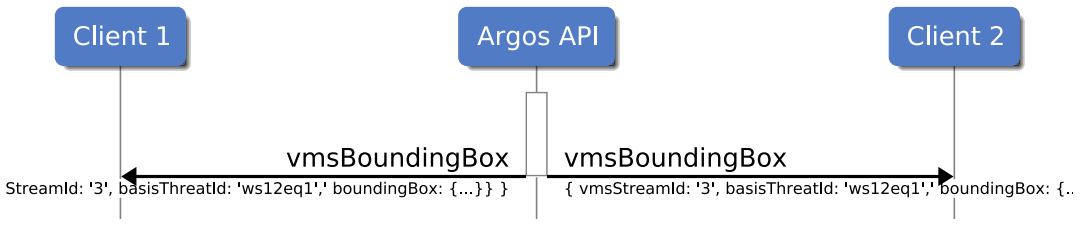

Figure 18.4: Example of vmsBoundingBox message communication from ARGOS API to a connected client when a new alarm zone has been created.

Information about the bounding box of a threat in a video stream. The bounding box is defined by pixels in the video stream.

19. Features

This part of the API is for informing the API client about the available features.

19.1 featureGetList

Pattern: Request/Response

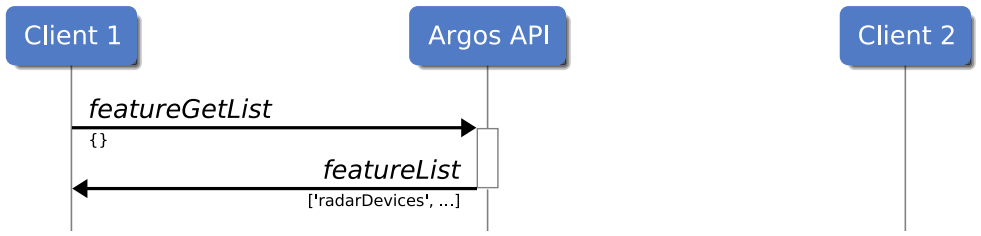

Figure 19.1: Example of featureGetList message communication between the client and the ARGOS API.

It is possible to fetch a list of the available backend features by sending the event «featureGetList». ARGOS will respond with the «featureList» message, which is a list of the available features.

Request: featureGetList
It is possible to send a request to get the list of features of the ARGOS backend.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/Empty.json`

Response: featureList
When a «featureGetList» request command is successfully registered it will respond with a «featureList» response.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FeatureList.json`

19.2 featureList

Pattern: Push

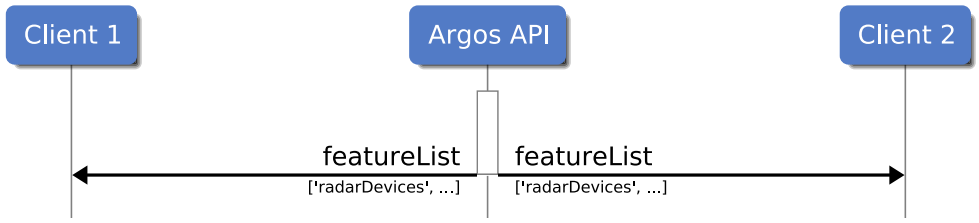

Figure 19.2: Example of featureList message communication from ARGOS API to a connected client.

During ARGOS startup or if an unforeseen error causes a restart of the internal feature management service a «featureList» message is pushed to all connected clients. The list describes all the available features of the ARGOS backend. This list can also be requested via «featureGetList» described in section 19.1 on the previous page.

For schema and samples, see file in ARGOS-schema archive:
`schema/messages/FeatureList.json`

20. Sapient

Setup of connections to a Sapient network is possible through this API. Individual devices in ARGOS can be assigned an endpoint in one or more Sapient networks. Connections are made from ARGOS and not from the actual device itself. Once a connection is established, relevant Uthreat information from the ARGOS stack is translated and sent via the Sapient protocol to the Sapient endpoint.

20.1 sapientConfigure

Pattern: Request/Response

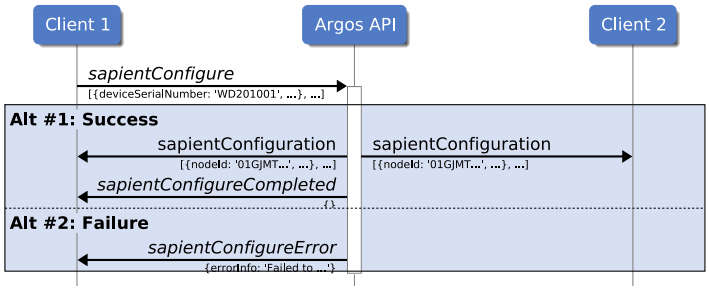

Figure 20.1: Example of sapientConfigure message communication between the client and the ARGOS API.

Use «sapientConfigure» to configure Sapient connections.

When using the «sapientConfigure» command, a response «sapientConfigureCompleted» message will be sent back as confirm. The operation causes a push «sapientConfiguration» message to be sent with information about the update. In case an error occurs during «sapientConfigure» processing, the «sapientConfigureError» message will be sent as response.

Request: sapientConfigure
The «sapientConfigure» request configures all Sapient connections.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/SapientConfigure.json`

When the «sapientConfigure» request has processed correctly the «sapientConfiguration» message will be pushed with information about the update.

Response: sapientConfigureCompleted
When «sapientConfigure» request command is successfully registered it will respond with the «sapientConfigureCompleted» response, or if an error occurs the «sapientConfigureError» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json` or
`schema/messages/Error.json`

20.2 sapientGetConfiguration

Pattern: Request/Response

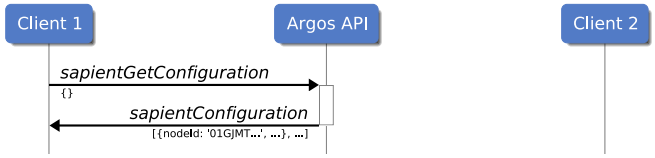

Figure 20.2: Example of sapientGetConfiguration message communication between the client and the ARGOS API.

Use «sapientGetConfiguration» to get configuration of all Sapient connections.

When using the «sapientGetConfiguration» command a response with «sapientConfiguration» will be sent back as response.

Request: sapientGetConfiguration
«sapientGetConfiguration» may be sent at any time.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json`

Response: sapientConfiguration
When «sapientGetConfiguration» request command is successfully registered it will respond with the «sapientConfiguration» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/SapientConfiguration.json`

20.3 sapientConfiguration

Pattern: Push

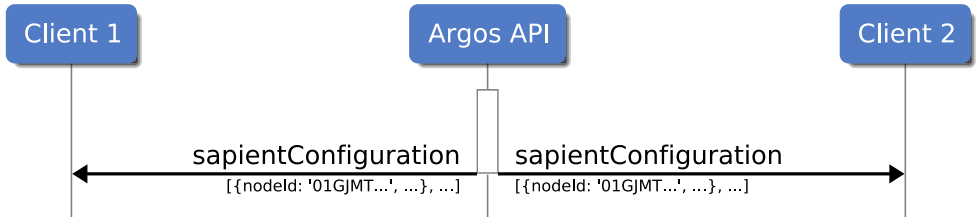

Figure 20.3: Example of sapientConfiguration message communication from ARGOS API to a connected client when a new alarm zone has been created.

When a «sapientConfigure» has been processed, a «sapientConfiguration» message

is pushed to all connected clients.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/SapientConfiguration.json`
During ARGOS startup or if an unforeseen error causes a restart of the internal Sapient service a «sapientConfiguration» message is pushed to all connected clients. This list can also be requested via «sapientGetConfiguration» described in section 20.2 on the preceding page.

20.4 sapientNodeGetList

Pattern: Request/Response

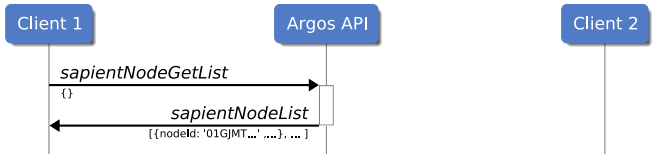

Figure 20.4: Example of sapientNodeGetList message communication between the client and the ARGOS API.

Use «sapientNodeGetList» to get a list of all Sapient nodes.

When using the «sapientNodeGetList» command a response with «sapientNodeList» will be sent back as response.

Request: sapientNodeGetList
«sapientNodeGetList» may be sent at any time.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json`

Response: sapientNodeList
When «sapientNodeGetList» request command is successfully registered it will respond with the «sapientNodeList» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/SapientNodeList.json`

20.5 sapientNodeList

Pattern: Push

During ARGOS startup or if an unforeseen error causes a restart of the internal Sapient service a «sapientNodeList» message is pushed to all connected clients. This list can also be requested via «sapientNodeGetList» described in section 20.4.

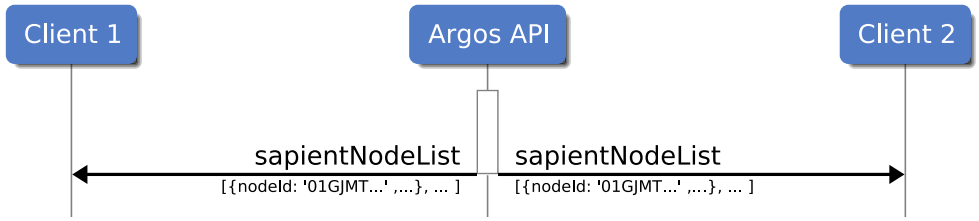

Figure 20.5: Example of sapientNodeList message communication from ARGOS API to a connected client when a new alarm zone has been created.

20.6 sapientNodeUpdated

Pattern: Push

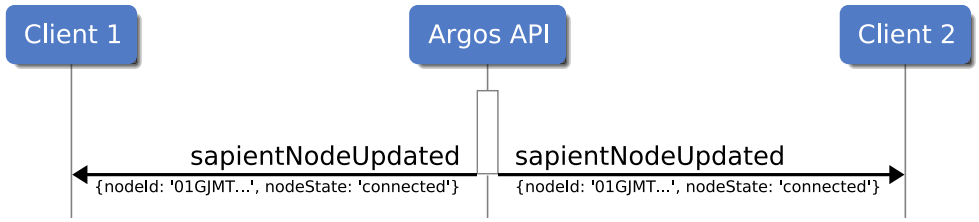

Figure 20.6: Example of sapientNodeUpdated message communication from ARGOS API to a connected client when a new alarm zone has been created.

When information regarding a Sapient node changes a «sapientNodeUpdated» is pushed to all connected clients.

21. TAK

Setup of connections to TAK servers is possible through this API. The individual connections can be configured as both input and output sources. Acting as an input source a connection filters relevant Cursor-on-Target (CoT) from the TAK server, and sends them into the Argos stack as Uthreat and device information. Acting as an output source a connection sends Uthreat and device information to a TAK server as CoT messages.

21.1 takConfigure

Pattern: Request/Response

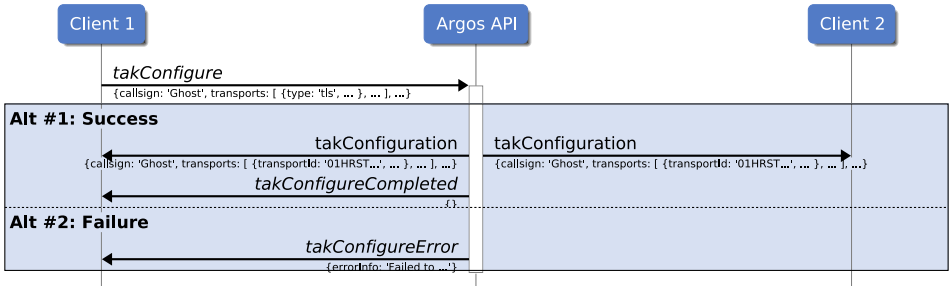

Figure 21.1: Example of takConfigure message communication between the client and the ARGOS API.

Use «takConfigure» to configure TAK server connections.

When using the «takConfigure» command, a response «takConfigureCompleted» message will be sent back as confirm. The operation causes a push «takConfiguration» message to be sent with information about the update. In case an error occurs during «takConfigure» processing, the «takConfigureError» message will be sent as response.

Request: takConfigure

The «takConfigure» request configures all TAK server connections.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/TakConfigure.json`

When the «takConfigure» request has processed correctly the «takConfiguration» message will be pushed with information about the update.

Response: takConfigureCompleted
When «takConfigure» request command is successfully registered it will respond with

the «takConfigureCompleted» response, or if an error occurs the «takConfigureError» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json` or
`schema/messages/Error.json`

21.2 takGetConfiguration

Pattern: Request/Response

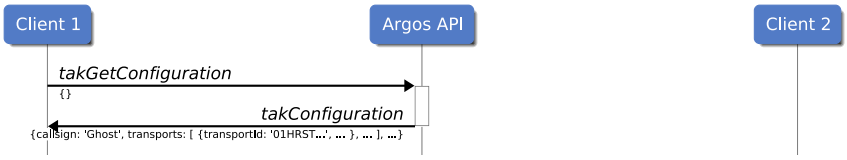

Figure 21.2: Example of takGetConfiguration message communication between the client and the ARGOS API.

Use «takGetConfiguration» to get configuration of all TAK server connections.

When using the «takGetConfiguration» command a response with «takConfiguration» will be sent back as response.

Request: takGetConfiguration
«takGetConfiguration» may be sent at any time.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json`

Response: takConfiguration
When «takGetConfiguration» request command is successfully registered it will respond with the «takConfiguration» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/TakConfiguration.json`

21.3 takConfiguration

Pattern: Push

When a «takConfigure» has been processed, a «takConfiguration» message is pushed to all connected clients.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/TakConfiguration.json`
During ARGOS startup or if an unforeseen error causes a restart of the internal TAK service a «takConfiguration» message is pushed to all connected clients. This list can

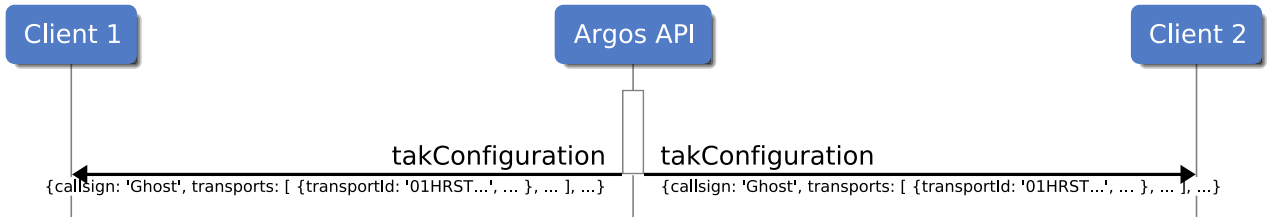

Figure 21.3: Example of takConfiguration message communication from ARGOS API to a connected client when a new alarm zone has been created.

also be requested via «takGetConfiguration» described in section 21.2 on the preceding page.

21.4 takTransportGetList

Pattern: Request/Response

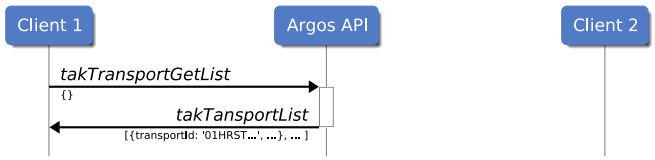

Figure 21.4: Example of takTransportGetList message communication between the client and the ARGOS API.

Use «takTansportGetList» to get a list of all TAK transports.

When using the «takTransportGetList» command a response with «takTansportList» will be sent back as response.

Request: takTransportGetList
«takTransportGetList» may be sent at any time.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/Empty.json`

Response: takTransportList
When «takTransportGetList» request command is successfully registered it will respond with the «takTransportList» response.

For schema and samples, see file in ARGOS - schema archive:
`schema/messages/TakTransportList.json`

21.5 takTransportList

Pattern: Push

During ARGOS startup or if an unforeseen error causes a restart of the internal TAK

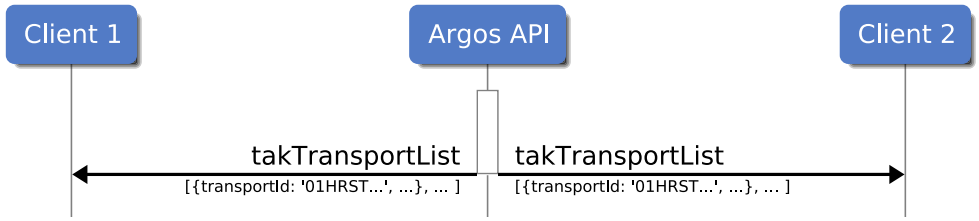

Figure 21.5: Example of takTransportList message communication from ARGOS API to a connected client when a new alarm zone has been created.

service a «takTransportList» message is pushed to all connected clients. This list can also be requested via «takTransportGetList» described in section 21.4 on the previous page.

21.6 takTransportUpdated

Pattern: Push

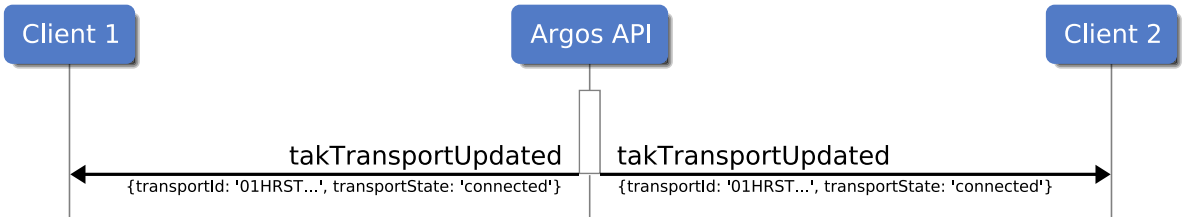

Figure 21.6: Example of takTransportUpdated message communication from ARGOS API to a connected client when a new alarm zone has been created.

When information regarding a TAK transport changes a «takTransportUpdated» is pushed to all connected clients.

MYDEFENCE

Counter Drone Technolo

MyDefence

Bouet Møllevej 5 • Nørresundby 9400 • Denmark

Telephone: +45 70 251 252

E-mail: sales@mydefence.dk / support@mydefence.dk