VDMA
Materials Handling
and Intralogistics

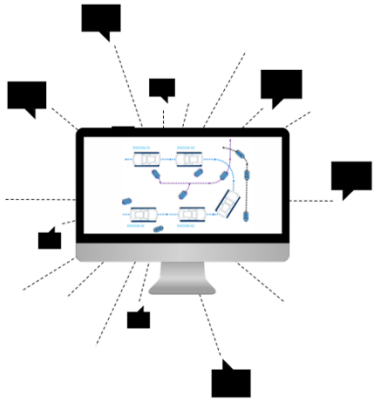
VDA
German Association
of the Automotive Industry

VDA Recommendation

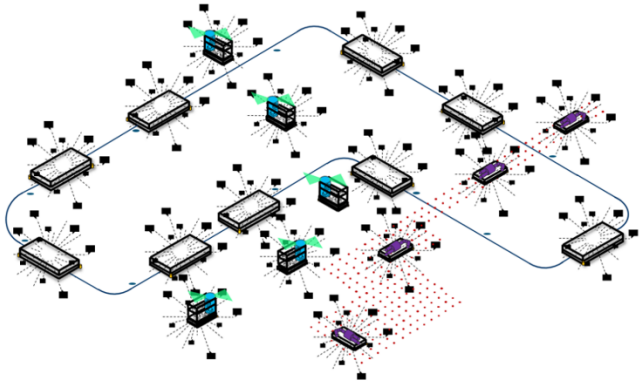
Interface for the Communication between Mobile Robots and a Fleet Control

VDA 5050

Version 3.0.0, March 2026



fleet control system



mobile robots

Disclaimer

The following explanations are intended to provide guidance for implementing an interface that enables communication between mobile robots and a fleet management system. They are intended to be freely accessible to all users and are non-binding. Any party choosing to apply these guidelines is responsible for ensuring their correct and appropriate use in each specific case.

Users must consider the applicable state of the art at the time the guidelines are applied. The use of these proposals does not relieve any party of responsibility for its own actions. These statements do not claim to be exhaustive, nor do they constitute an authoritative interpretation of existing laws. They do not replace the need to review and comply with relevant policies, legislation, or regulations. In addition, the specific characteristics of the respective products and their various potential applications must be considered.

All users act at their own risk. Any liability on the part of the VDA and VDMA or any individuals involved in the development or application of these proposals is excluded. If you identify any inaccuracies in the application of these proposals or potential risks of misinterpretation, please notify the VDA immediately so that any necessary corrections can be made.

Publisher Verband der Automobilindustrie e. V. (VDA) Behrenstraße 35, 10117 Berlin, Germany www.vda.de

Disclaimer

The VDA Recommendations are recommendations that may be freely adopted by anyone. Users are responsible for correct implementation of the recommendations as required on a case-by-case basis.

The recommendations take into account the prevailing technology at the time of publication. Use of the VDA Recommendations does not absolve anyone from responsibility for his/her own actions, and all users act at their own risk. Liability of VDA and those involved in drafting of VDA Recommendations is excluded.

Table of contents

0 Foreword 6

1 Introduction 6

2 Scope 6

3 Definitions 7

3.1 Mobile Robot 7
3.2 Moving 7
3.3 Driving 7
3.4 Automatic Driving 7
3.5 Manual Driving 8
3.6 Line-guided mobile robot 8
3.7 Freely navigating mobile robot 8

4 Transport protocol 8

4.1 Connection handling, security and QoS 8
4.2 Topic levels 8
4.3 Topics for communication 9

5 Process and content of communication 10

5.1 General 10
5.2 Implementation Phase 11
5.3 Functions of the fleet control 11
5.4 Functions of the mobile robots 11

6 Protocol specification 12

6.1 Order 12
6.2 Actions 23
6.3 Maps 39
6.4 Zones 42
6.5 Connection 51
6.6 State 51
6.7 Visualization 64
6.8 Sharing of planned paths for freely navigating mobile robots 64
6.9 Request/response mechanism 65
6.10 Factsheet 66

7 Message specification 67
7.1 Symbols of the tables and meaning of formatting 67
7.2 Protocol header 68
7.3 Implementation of the order message 68
7.4 Implementation of the instantAction message 75
7.5 Implementation of the response message 75
7.6 Implementation of the zoneSet message 76
7.7 Implementation of the connection message 78
7.8 Implementation of the state message 79
7.9 Implementation of the visualization message 91
7.10 Implementation of the factsheet message 92

Bibliography 102

List of Figures

Figure 1: Structure of the information flow 10
Figure 2: Graph representation in fleet control and graph transmitted in orders 12
Figure 3: Procedure for expanding the driving route “Horizon” 13
Figure 4: Pseudocode of an order 14
Figure 5: Pseudocode of an order update. Note the change of the orderUpdateId 14
Figure 6: Regular update process - order extension 15
Figure 7: Order update with additional stitching node (e.g., to execute new actions on decision point) 16
Figure 8: The process of accepting an order or order update 17
Figure 9: Expected behavior after a cancelOrder 19
Figure 10: Edges with a corridor attribute that defines the left and right boundaries within which a mobile robot is allowed to deviate from its predefined trajectory to avoid obstacles 23
Figure 11: Handling multiple actions 25
Figure 12: Coordinate system with sample mobile robot and orientation 39
Figure 13: Communication required between fleet control, mobile robot and map server to download, enable, and delete a map 40
Figure 14: Depiction of a mobile robot entering a zone based on its contour (left) and a loaded mobile robot with corresponding extended bounding box exiting a zone (right) 42
Figure 15: Depiction of a mobile robot entering a zone based on its kinematic center (left) and a loaded mobile robot exiting a zone based on its kinematic center (right) 44
Figure 16: Zone request behavior for a RELEASE zone 48
Figure 17: Zone request behavior for a COORDINATED_REPLANNING zone 49
Figure 18: Order information provided by the state topic. Only the ID of the last node and the remaining nodes and edges are transmitted 52
Figure 19: Depiction of nodeStates, edgeStates, and actionStates during order handling 53
Figure 20: allowedDeviation ellipse 54
Figure 21: All possible status transitions for actionStates 63
Figure 22: Request lifecycle: request states and logic of possible transitions 65

List of tables

Table 1: Explanation of suggested MQTT topic levels 9
Table 2: Topics for communication between fleet control and mobile robotProcess and content of communication 10
Table 3: Definition of action blocking types dependent on driving and parallel execution 24
Table 4: Predefined actions and their scope (instant, node, edge, zone) 32
Table 5: Expected behavior in action states of predefined actions 38
Table 6: Contour-based zone types and their parameters 44
Table 7: Kinematic center-based zone types and their parameters 45
Table 8: Interaction matrix for zones 50
Table 9: Predefined error types 58
Table 10: Operating modes of the mobile robot 60
Table 11: Overview of operating modes and their implications 60
Table 12: Feasible values for the actionStatus field 61
Table 13: Examples for possible action state transitions 62
Table 14: Symbols of the tables and meaning of formatting 67

0 Foreword

The specification for this interface has been jointly developed by the Verband der Automobilindustrie e. V. (VDA) and the VDMA e. V. (Mechanical Engineering Industry Association). The VDA represents the German automotive sector, including OEMs and Tier-1/Tier-n suppliers, and contributes its expertise in vehicle architectures, system integration, and safety-critical communication. The VDMA represents companies across the European mechanical and plant engineering industry and brings extensive knowledge in automation technology, machinery interoperability, and production system standardization. Both organizations collaborate to ensure that the interface specification reflects current engineering requirements, supports robust and scalable system integration, and enables consistent data exchange across heterogeneous environments. Their joint development process emphasizes harmonized communication models, compatibility with established industrial standards, and long-term maintainability of cross-domain interfaces. This cooperation ensures that the resulting specification can be reliably implemented in automotive, machinery, and mixed-industry applications, supporting high interoperability, operational safety, and future-proof system architectures. The Institute for Material Handling and Logistics (IFL) at Karlsruhe Institute of Technology (KIT) is part of the department of mechanical engineering and focuses on combining research, teaching, and industrial application. Its interdisciplinary team works on future logistics challenges, including material flow analysis, automation, robotics, digitalization, AI, sustainability, and system design. The Institute has been commissioned by the VDA and the VDMA to oversee the development of the VDA 5050. It contributes to this process by taking the lead in development, supporting issue review, and managing the official GitHub repository.

1 Introduction

This recommendation describes the communication interface for exchanging information between central fleet control and mobile robots. The objective of this recommendation is to support the integration and efficient operation of mobile robot fleets under the supervision of a centralized fleet control system. This is achieved through the implementation of a standardized, vendor neutral communication interface that ensures interoperability between the fleet control system and individual mobile robots. Various national technical guidelines and legal frameworks may offer general orientation in this context. They could provide indicative information on aspects such as planning, operation, safety, or coordination of automated systems. In addition, national standards and regulatory provisions may help ensure that technical processes and terminology are considered within a consistent overall framework. The recommendation uses a semantic versioning schema. Major version changes (x.0.0) typically involve breaking changes, such as the introduction of new non optional fields. Minor version changes (3.x.0) generally introduce new features, for example the addition of an optional parameter for visualization. Patch version changes (3.0.x) usually address smaller corrections, such as fixing typographical errors in the documentation. Stakeholders are invited to submit proposals for modifications or enhancements to the interface. Such proposals shall be submitted via the GitHub repository at: https://github.com/vda5050/vda5050.

2 Scope

This document describes a standardized and vendor-neutral communication interface between a fleet control system and mobile robots. Its purpose is to provide a common reference that supports interoperability in environments where multiple mobile robots operate under the coordination of a fleet control system. The use of this specification is optional and non-binding, and its application is at the discretion of the respective stakeholders.

The objectives of this specification are

- to reduce complexity when connecting mobile robots to a fleet control system.
- to enable the coordinated operation of heterogeneous mobile robot fleets from different manufacturers within a shared physical environment.
- to provide a generic and domain independent set of interface definitions applicable to mobile robots with varying navigation principles, physical dimensions, load handling or manipulation capabilities, and autonomy levels.

This specification does not address the following topics:

- Safety Requirements:
  This document does not define functional, operational, or system safety requirements and shall not be regarded or applied as a safety standard.
- Traffic Management Logic:
  Strategies, algorithms, or decision making processes for traffic coordination (e.g., routing, prioritization, congestion handling, or deadlock resolution) are not included.
- Other Communication Interfaces:
  Interfaces unrelated to the communication between a fleet control system and mobile robots are excluded, such as interfaces to peripheral equipment, infrastructure components, or external IT systems.
- Project Coordination and Implementation Procedures:
  Project management activities, integration methodologies, commissioning workflows, validation and acceptance procedures, and similar organizational processes are not covered.
- Operational Responsibilities:
  This document does not allocate responsibilities among operators, system integrators, vehicle manufacturers, or fleet control providers with respect to planning, operation, maintenance, or safety.
- Cybersecurity Measures:
  Mechanisms, technologies, or processes for secure communication or data protection are not specified.

3 Definitions

The following terms and definitions apply for the purposes of this document. Terms that are not officially defined by standardization organizations may be interpreted differently in other contexts.

3.1 Mobile Robot

A driverless system for material transport primarily in operational settings, controlled by automation independently of their level of autonomy [Source ISO 3691-4].

3.2 Moving

State in which a mobile robot or any of its components undergoes a change in spatial position or orientation, including movement of wheels, load handling devices, or the robot body.

3.3 Driving

Operating state in which the mobile robot has a non zero translational and/or rotational velocity.

3.4 Automatic Driving

Driving state in which the mobile robot operates without human intervention.

3.5 Manual Driving

Driving state in which the mobile robot operates under direct human control.

3.6 Line-guided mobile robot

Mobile robots that follow predefined trajectories. Predefined trajectories are sent by fleet control as part of the order or defined on the robot, either explicitly or implicitly as the direct connection between nodes.

3.7 Freely navigating mobile robot

Mobile robots that plan their own trajectories. If fleet control sends a trajectory within the order, the robot shall follow this trajectory.

4 Transport protocol

Communication is expected to be done via wireless networks, considering the effects of connection failures and potential loss of messages.

The message protocol is Message Queuing Telemetry Transport (MQTT), which is to be used in combination with a JSON format. MQTT 3.1.1 is the minimum required version for compatibility. MQTT allows the distribution of messages to subchannels, which are called “topics”. Participants in the MQTT network subscribe to these topics and receive information that concerns them.

The JSON format allows for future extensions of the protocol with additional parameters as well as validation against schemas.

4.1 Connection handling, security and QoS

The MQTT protocol provides the option of setting a last will message for a client. If the client disconnects unexpectedly for any reason, the last will is distributed by the broker to other subscribed clients. The use of this feature is described in Section 6.5 Connection.

If the mobile robot disconnects from the broker, it keeps all the order information and fulfills the order up to the last released node.

To reduce the communication overhead, the MQTT QoS level 0 (Best Effort) shall be used for the topics order, instantActions, state, factsheet, zoneSet, responses and visualization. QoS level 1 (At Least Once) shall be used for the topic connection.

Protocol security needs to be taken into account by broker configuration, but is not addressed within this guideline.

4.2 Topic levels

The MQTT topic structure is not strictly defined due to the mandatory topic structure of cloud providers. For a cloud-based MQTT broker the topic structure might have to be adapted individually, but it should roughly follow the proposed structure. The topic names defined in the following sections are mandatory.

For a local broker the MQTT topic levels are suggested as followed:

interfaceName/majorVersion/manufacturer/serialNumber/topic

Example:

`vda5050/v3/KIT/0001/order`

MQTT Topic Level	Data type	Description
interfaceName	string	Name of the used interface
majorVersion	string	Major version number of the VDA 5050 recommendation, preceded by “v”
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Unique mobile robot serial number consisting of the following characters: A-Z a-z 0-9 _ . : -
topic	string	Topic (e.g., order or state) see Section 4.4 Topics for Communication

Table 1: Explanation of suggested MQTT topic levels

Since the / character is used to define topic hierarchies, it shall not be used in any of the aforementioned fields. Wildcard characters + and # as well as the character $ that is reserved for broker internal topics should not be used either.

4.3 Topics for communication

The protocol uses the following topics for information exchange between fleet control and mobile robots.

Topic name	Published by	Subscribed by	Used for	Implementation	Schema
order	fleet control	mobile robot	Communication of orders	mandatory	order.schema
instantActions	fleet control	mobile robot	Communication of the actions that are to be executed immediately	mandatory	instantActions.schema
state	mobile robot	fleet control	Communication of the mobile robot state	mandatory	state.schema
visualization	mobile robot	visualization systems	High frequency communication of position and planned path	optional	visualization.schema
connection	broker / mobile robot	fleet control	Indicates when mobile robot connection is lost. Not to be used by fleet control for checking the mobile robot health, added for an MQTT protocol level check of connection	mandatory	connection.schema

Topic name	Published by	Subscribed by	Used for	Implementation	Schema
factsheet	mobile robot	fleet control	Parameters or vendor-specific information to assist set-up of the mobile robot in fleet control	mandatory	factsheet.schema
zoneSet	fleet control	mobile robot	Transfer of zone sets from fleet control to the mobile robot	optional	zoneSet.schema
responses	fleet control	mobile robot	Fleet control's responses to requests from within the mobile robot's state	optional	responses.schema

Table 2: Topics for communication between fleet control and mobile robotProcess and content of communication

5 Process and content of communication

5.1 General

There are at least the following participants for the operation of driverless transport system:

- The operator of the DTS provides basic information
- The fleet control organizes and manages the operation
- The mobile robot carries out the orders

Figure 1 describes the communication content during the operational phase. During implementation or modification, the mobile robot and the fleet control are manually configured.

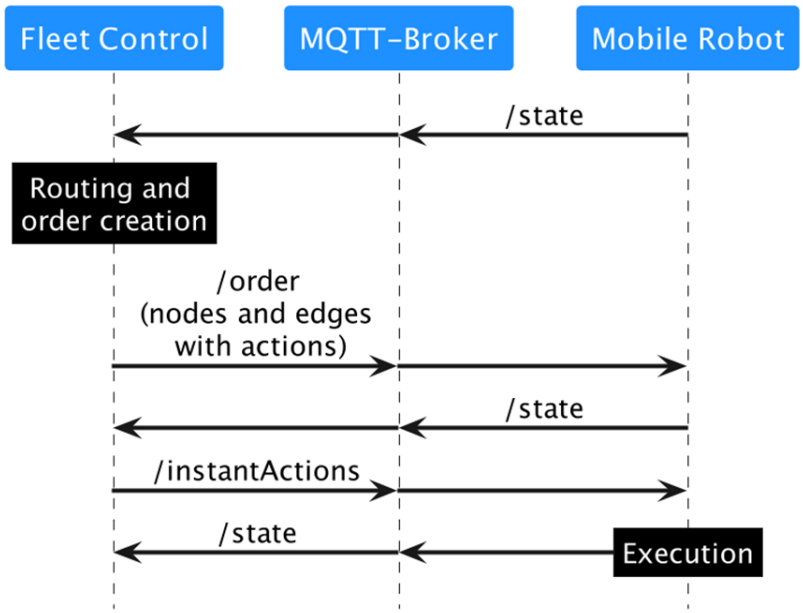


Figure 1: Structure of the information flow

5.2 Implementation Phase

During the implementation phase, the DTS consisting of fleet control and mobile robots is set up. The necessary framework conditions are defined by the operator and the required information is either entered manually by them or stored in the fleet control by importing from other systems. Essentially, this concerns the following content:

- Definition of routes: Using the Layout Interchange Format (LIF), routes can be imported to the fleet control. The LIF is a file format of track layouts for exchange between the integrator of the driverless transport mobile robots and a (third-party) fleet control system (LIF – Layout Interchange Format, VDMA 2024-03). Alternatively, routes can also be implemented manually in the fleet control by the operator. Routes can be one-way streets, restricted for certain mobile robot groups (based on the size ratios), etc.
- Route network configuration: Within the routes, stations for loading and unloading, battery charging stations, peripheral environments (gates, elevators, barriers), waiting positions, buffer stations, etc. are defined.
- Mobile robot configuration: The physical properties of a mobile robot (size, available load carrier mounts, etc.) are stored by the operator. The mobile robot shall communicate this information via the topic factsheet in a specific way that is defined in Section 7.10 Implementation of the factsheet message of this document.

The configuration of routes and the route network described above are not part of this document. They form the basis for enabling order control and driving course assignment by the fleet control based on this information and the transport requirements to be completed. The resulting orders to be executed by the robotic fleet are transferred to the individual mobile robots via MQTT. The mobile robot then continuously reports its status to the fleet control in parallel with the execution of the order, also using MQTT.

5.3 Functions of the fleet control

The fleet control system performs, at minimum, the following functions:

- Assignment of orders to the mobile robots
- Route calculation and guidance of line-guided mobile robots (taking into account the limitations of the individual physical properties of each mobile robot, e.g., size, maneuverability, etc.)
- Detection and resolution of blockages (“deadlocks”)
- Energy management: Charging orders can interrupt transfer orders
- Traffic control: Buffer routes and waiting positions
- (Temporary) changes in the environment, such as freeing certain areas or changing the maximum speed
- Communication with peripheral systems such as doors, gates, elevators, etc.
- Detection and resolution of communication errors

5.4 Functions of the mobile robots

Each mobile robot shall perform the following functions:

- Localization
- Execution of associated routes (line-guided or freely navigating)
- Execution of actions
- Continuous transmission of its status

6 Protocol specification

The following section describes the details of the communication protocol. The protocol specifies the communication between the fleet control and the mobile robot.

6.1 Order

The topic order is the MQTT topic via which the mobile robot receives an order, containing instructions for the robot to move or execute actions.

6.1.1 Concept and logic

The core of a transport order is a node-edge-graph segment defining the route to be travelled. The mobile robot is expected to traverse the nodes and edges to fulfill the order. The full graph of all connected nodes and edges is held by fleet control. It may contain restrictions, e.g., which mobile robot is allowed to traverse which edge. These restrictions will not be communicated to the mobile robot. The fleet control only includes edges in an order which the concerning mobile robot is allowed to traverse.

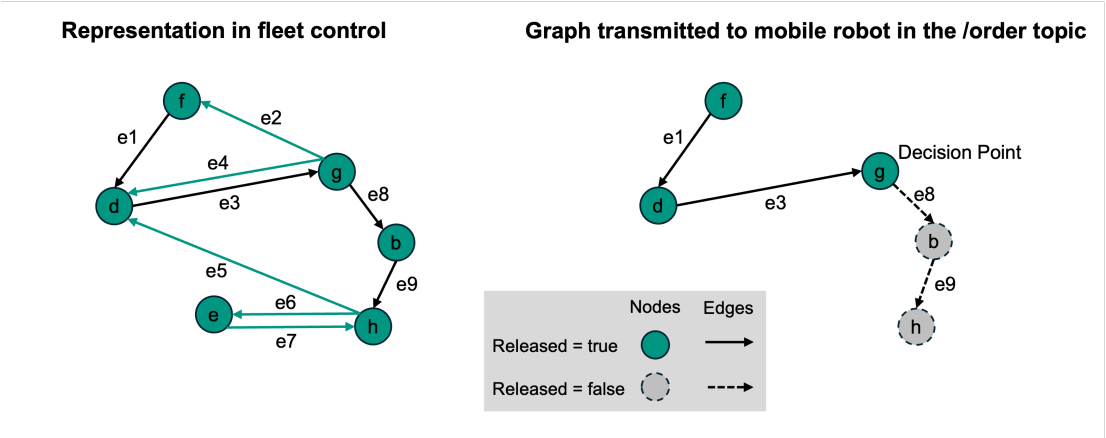


Figure 2: Graph representation in fleet control and graph transmitted in orders

The nodes and edges are passed as two lists in the order message. The order of the nodes and edges within those lists also governs the sequence in which the nodes and edges shall be traversed. The ‘sequenceId’ is shared between nodes and edges and defines the sequence of traversal. The first node has a `sequenceId` of 0, the first edge has a `sequenceId` of 1, the second node has a `sequenceId` of 2, etc. An edge with `sequenceId` n connects the nodes with `sequenceId` n-1 and n+1. The `sequenceId` shall be continuous within an order.

For a valid order, there shall be at least one node and the number of edges shall be equal to the number of nodes minus one.

The first node of an order (`sequenceId` = 0) shall be trivially reachable for the mobile robot and always be released. This means either that the mobile robot is already standing on the node, or that the mobile robot is in the node’s deviation range. As such, the first node shall not be reported in the `nodeStates`.

Nodes and edges both have a boolean attribute `released`. If a node or edge is released, the mobile robot is expected to traverse it. If a node or edge is not released, the mobile robot shall not traverse it.

An edge can be released only if both the start and the end node of the edge are released.

After an unreleased edge, no released nodes or edges can follow in the sequence.

The set of released nodes and edges are called the “base”. The set of unreleased nodes and edges are called the “horizon”.

It is valid to send an order without a horizon.

An order message does not necessarily describe the full transport order. For traffic control and to accommodate resource constrained mobile robots, the full transport order (which might consist of many nodes and edges) can be split up into many sub-orders, which are connected via their `orderId` and `orderUpdateId`. The process of updating an order is described in the next section.

6.1.2 Orders and order update

To support traffic management, fleet control can split the path communicated via order into two parts:

- *"Base":* This is the defined route that the mobile robot is allowed to travel. All nodes and edges of the base route have already been released by the fleet control for the mobile robot. The last node of the base is called decision point.
- *"Horizon":* This is the route currently planned by fleet control for the mobile robot to travel after the decision point. The horizon route has not yet been released by the fleet control.

The mobile robot shall stop at the decision point if no further nodes and edges are added to the base. In order to ensure a fluent movement, the fleet control should extend the base before the mobile robot reaches the decision point, if the traffic situation allows for it.

Since MQTT is an asynchronous protocol and transmission via wireless networks is not reliable, the base cannot be changed. The fleet control shall therefore assume that the base has already been executed by the mobile robot. A later section describes a procedure to cancel an order, but this is also considered unreliable due to the communication limitations mentioned above.

The fleet control can change the horizon by sending an updated route to the mobile robot which includes the changed list of nodes and edges. The procedure for changing the horizon route is shown in Figure 3.

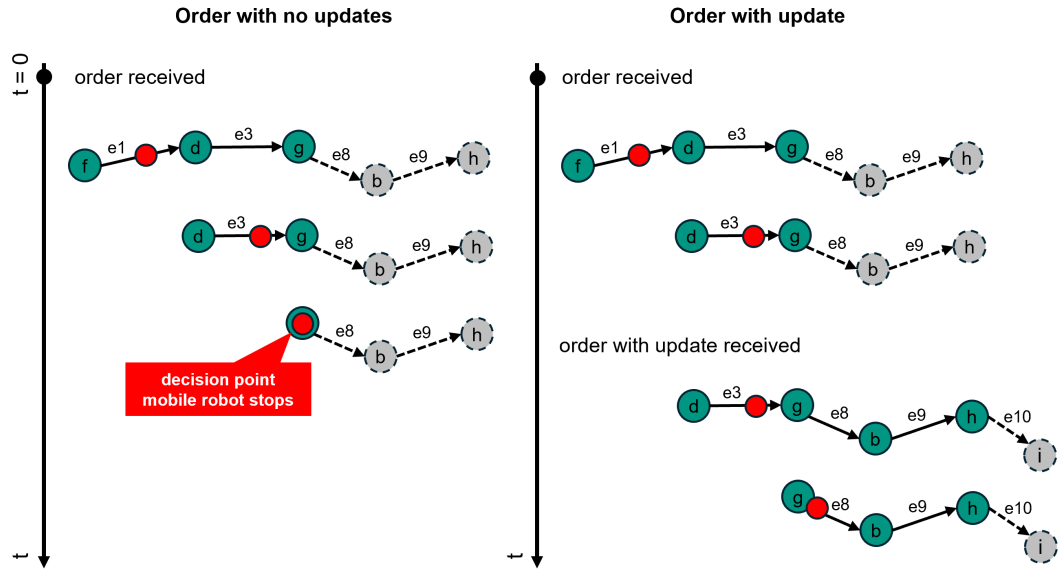


Figure 3: Procedure for expanding the driving route "Horizon"

In Figure 3, an initial order is first sent by the fleet control at time t = 0. Figure 4 shows the pseudocode of a possible order. For the sake of readability, a complete JSON example has been omitted here.

```
{
    orderId: "1234"
    orderUpdateId:0,
    nodes: [
         f {released: true},
         d {released: true},
         g {released: true},
         b {released: false},
         h {released: false}
    ],
    edges: [
        e1 {released: true},
        e3 {released: true},
        e8 {released: false},
        e9 {released: false}
    ]
}
```

Figure 4: Pseudocode of an order

At a later point in time, the order is extended by sending an order update (see pseudocode in Figure 5). Note that the `orderUpdateId` is incremented and that the first node of the order update corresponds to the last base node of the previous order message, the stitching node. The other nodes and edges from the previous base are not resent.

This ensures that the mobile robot can also perform the order update, i.e., that the first node of the order update is reachable by executing the edges already known to the mobile robot.

```
{
    orderId: "1234",
    orderUpdateId: 1,
    nodes: [
        g {released: true},
        b {released: true},
        h {released: true},
        i {released: false}
    ],
    edges: [
        e8 {released: true},
        e9 {released: true},
        e10 {released: false}
    ]
}
```

Figure 5: Pseudocode of an order update. Note the change of the orderUpdateId

This also aids in the event that an order update is lost (e.g., due to an unreliable wireless network). The mobile robot can always check that the last known base node has the same `nodeId` (and `sequenceId`) as the first node of a new order update.

Also note that node g is the only base node that is sent again. Since the base cannot be changed, a retransmission of nodes f and d is not valid.

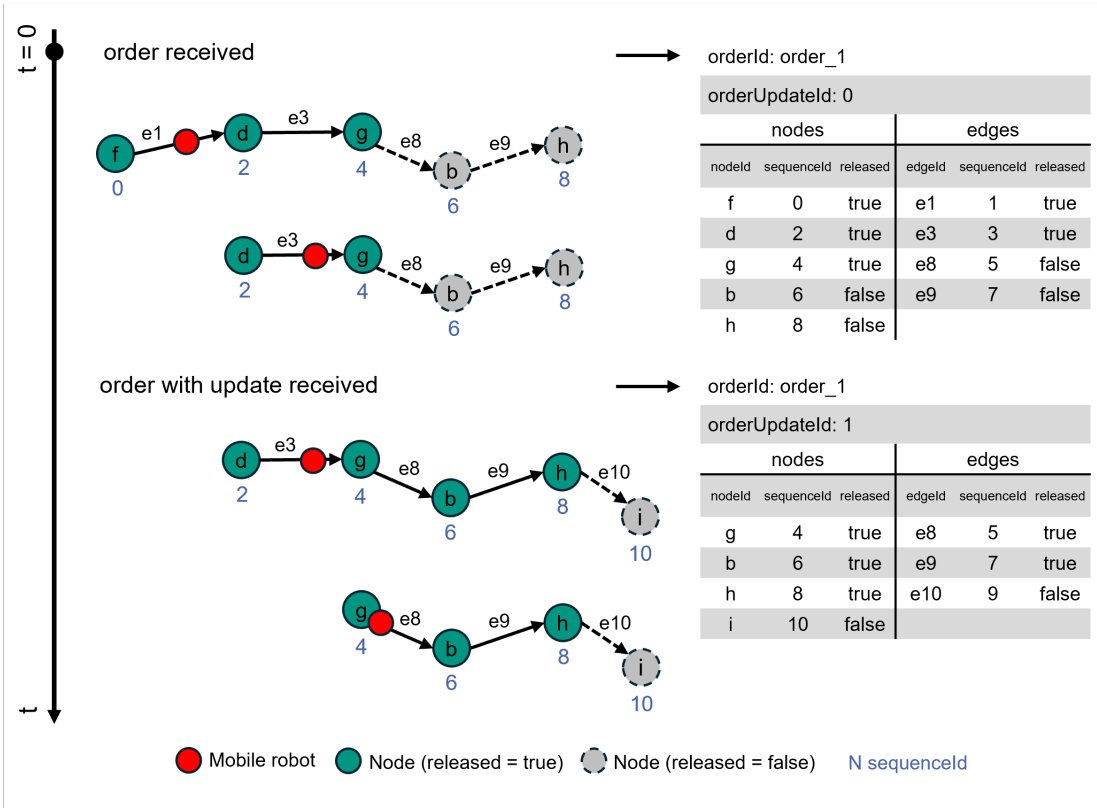


Figure 6: Regular update process - order extension

Figure 6 describes how an order should be extended. It shows the information that is currently available on the mobile robot. The `orderId` stays the same and the `orderUpdateId` is incremented.

It is important that the contents of the decision point (node g in Figure 6) are not changed. This means actions, deviation range, etc., shall be resent (see Figure 7, `orderUpdateId` 1). In order to release actions for the mobile robot to execute on a node it is already positioned on through an order update, the fleet control shall re-send this node once with all meta-data (including potentially already 'FINISHED'/'RUNNING' actions) from the previous order update, which will not be executed again by the mobile robot, and then add a node with the now newly released actions to be executed with this order update. This node can have the same `nodeId` as the decision node or a different `nodeId` but the same position as the decision node. The `sequenceId` of the new node is always the `sequenceId` of the decision node plus 2.

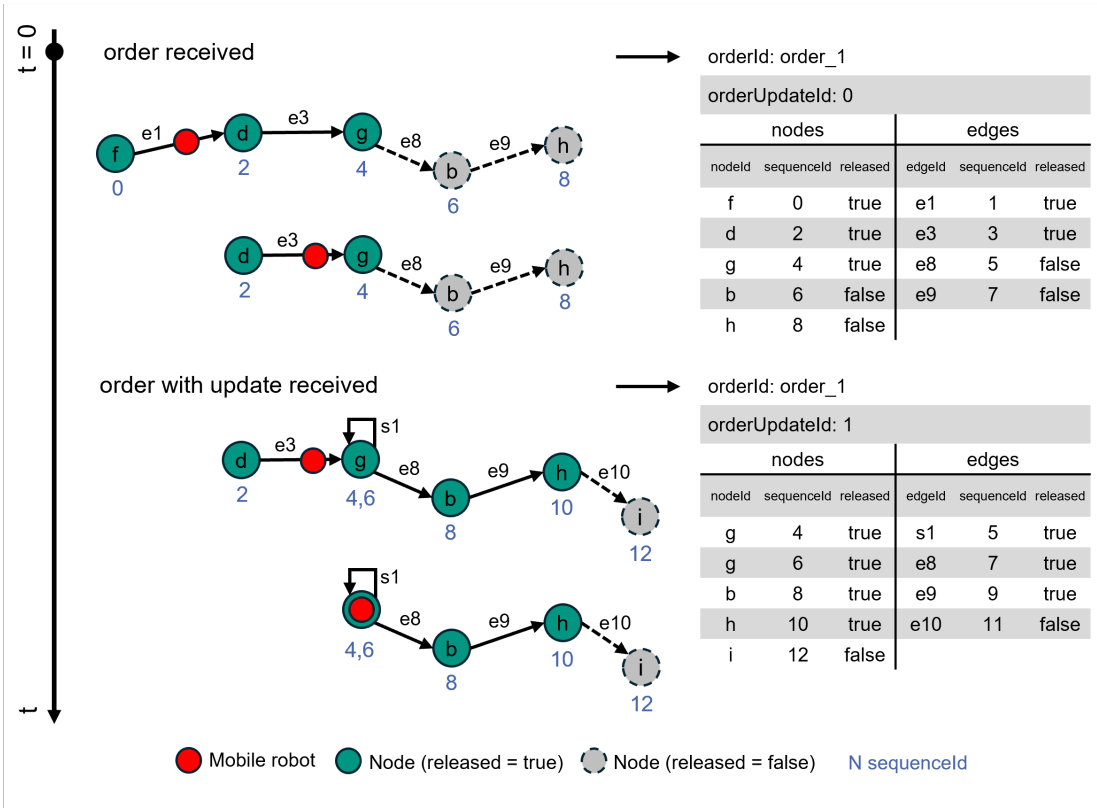


Figure 7: Order update with additional stitching node (e.g., to execute new actions on decision point)

The horizon may be modified or deleted entirely with any order update, or the base may be extended in a way different from the previous horizon.

Once a `sequenceId` is assigned and the node is released, it does not change with order updates (see Figure 6).

Figure 8 describes the process of accepting an order or order update.

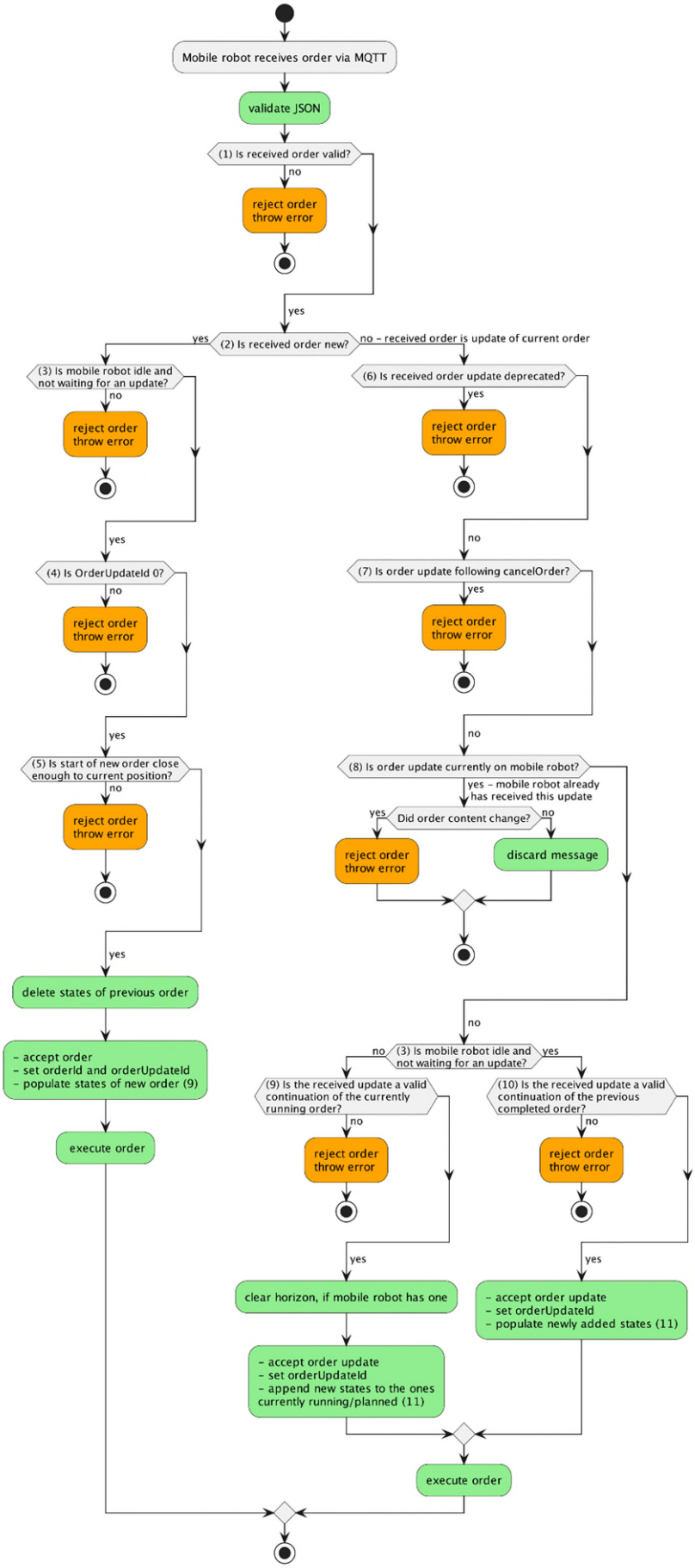


Figure 8: The process of accepting an order or order update

1) **Is received order valid?**: All formatting and JSON data types are correct?

2) **Is received order new or an update of the current order?**: Is `orderId` of the received order different to `orderId` of order the mobile robot currently holds?

3) **Is mobile robot idle and not waiting for an update?**: Is the mobile robot in an idle state according to 6.6.8 Idle state of the mobile robot and not waiting for an update? Since nodes and edges and the corresponding action states of the order horizon are also included inside the state, the mobile robot might still have a horizon and therefore is waiting for an update and executing an order.

4) **Is OrderUpdateId 0?**: Is the `orderUpdateId` of the new order 0?

5) **Is start of new order close enough to current position?**: Is the mobile robot already standing on the node, or is it in the node's deviation range (6.1.1 Concept and logic)?

6) **Is received order update deprecated?**: Is `orderUpdateId` less than or equal to one currently on the mobile robot?

7) **Is order update following cancelOrder?**: No further order updates to the cancelled order shall be sent by the fleet control or accepted by the mobile robot.

8) **Is received order update currently on mobile robot?**: Is `orderUpdateId` equal to the one currently on the mobile robot?

9) **Is the received update a valid continuation of the currently still running order?**: Is the first node of the received order the current decision point according to the order update chapter? The mobile robot is still moving or executing actions related to the base released in previous order updates or still has a horizon and is therefore waiting for a continuation of the order. In this case, the order update is only accepted if the first node of the new base is equal to the last node of the previous base.

10) **Is the received update a valid continuation of the previously completed order?**: Is the first node of the received order the current decision point according to the order update chapter? The mobile robot is not executing any actions anymore neither is it waiting for a continuation of the order (meaning that it has completed its base with all related actions and does not have a horizon). In this case, the order update is only accepted if the first node of the new base is equal to the last node of the previous base.

11) **Populate/append** new states to the `actionStates`/`nodeStates`/`edgeStates`.

6.1.2.1 Finishing an order

After the mobile robot has traversed the last node of an order and has finished all order related movement and actions, it is idle and shall be ready to receive a new order (see 6.6.8 Idle state of the mobile robot).

6.1.3 Order cancellation

Fleet control can cancel an active order using the instantAction `cancelOrder`.

Fleet control can optionally pass an `orderId` to reference which order shall be canceled. After receiving the instantAction `cancelOrder`, the mobile robot shall attempt to stop as soon as possible. For line-guided mobile robots, this could be the next feasible node. A freely navigating mobile robot shall stop as soon as possible, not merely at the next node.

If there are actions in the `actionStates` scheduled, these actions shall be cancelled and report 'FAILED' in their `actionState`. If there are actions in the `actionStates` running, those actions should be cancelled and also be reported as 'FAILED'. If the action cannot be cancelled, the `actionState` of that action should reflect that by reporting 'RUNNING' while it

is running, and after that the respective state (‘FINISHED’, if successful and ‘FAILED’, if not). While there are running actions in the `actionStates`, the cancelOrder action shall report ‘RUNNING’ until all actions are cancelled/finished. Actions that cannot be cancelled (cancelAllowed = false) shall be finished. After all movement of the mobile robot and all of the actions in the `actionStates` are stopped, the `cancelOrder` action status shall report ‘FINISHED’. The mobile robot shall then be idle and ready to receive new orders.

The `orderId` and `orderUpdateId` are kept.

Figure 9 shows the expected behavior for different mobile robot capabilities.

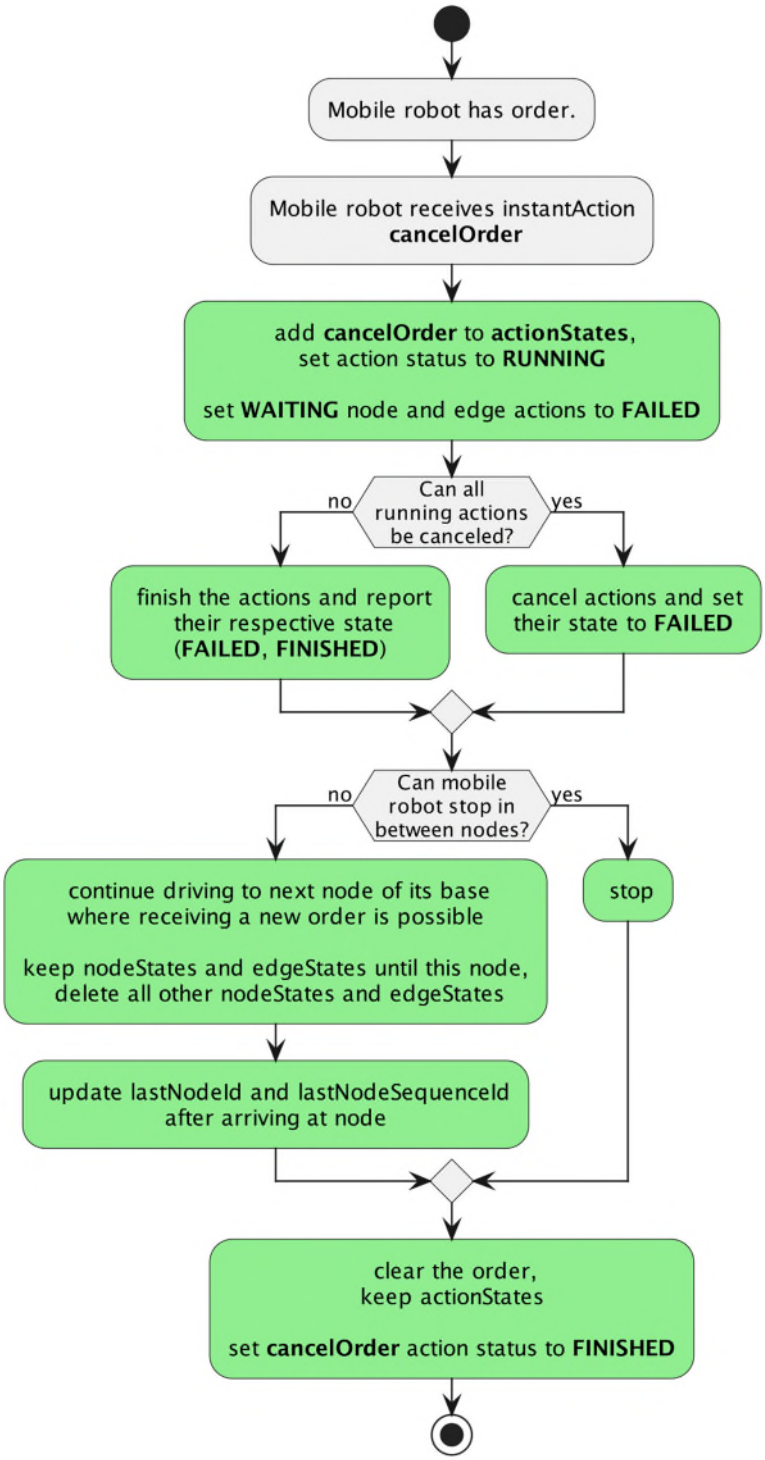


Figure 9: Expected behavior after a cancelOrder

6.1.3.1 Receiving a new order after cancellation

After the cancellation of an order, the mobile robot is idle and shall be ready to receive a new order. No further order updates to the cancelled order shall be sent by the fleet control. If the mobile robot receives an order update it shall report an error of type ‘ORDER_UPDATE_FOLLOWING_CANCEL’ and level ‘WARNING’.

In the case of a mobile robot that can only localize itself on a node, the new order shall begin on the node the mobile robot is now standing on (see also Figure 4).

In case of a mobile robot that can stop in between nodes, fleet control can decide how to start the next order. The mobile robot shall accept both methods.

There are two options:

- The first node of the new order is a temporary node that is positioned at the mobile robot’s current position. The mobile robot shall then recognize that this node is trivially reachable and accept the order.
- The first node of the new order is the last traversed node of the previous order. The allowed deviation of this node is set large enough to ensure that the mobile robot is within this range. Thus, the mobile robot shall immediately treat this node as traversed and accept the order.

6.1.3.2 Receiving a cancelOrder action when mobile robot is idle

If the mobile robot receives a `cancelOrder` instant action but the mobile robot is currently idle, or the `orderId` specified in the action does not match the `orderId` of the mobile robot’s currently active order, the `cancelOrder` action shall be reported as ‘FAILED’.

The mobile robot shall report an error of type ‘NO_ORDER_TO_CANCEL’ with the level set to ‘WARNING’. The `actionId` of the `instantAction` shall be passed as an `errorReference`.

6.1.4 Order rejection

There are several scenarios, when an order shall be rejected. These scenarios are shown in Figure 8 and described below.

6.1.4.1 Mobile robot receives a malformed order

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer.
2. The mobile robot shall report an error of type ‘VALIDATION_FAILURE’ and level ’WARNING‘
3. The warning shall be reported until the mobile robot has accepted a new order.

6.1.4.2 Mobile robot receives an order with optional fields it cannot use

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer
2. The mobile robot shall report an error of type ‘UNSUPPORTED_PARAMETER’ with level ‘CRITICAL’ and the erroneous fields as errorReferences
3. The error shall be reported until the mobile robot has accepted a new order.

6.1.4.3 Mobile robot receives an order with actions it cannot perform

Example:

- lifting height higher than maximum lifting height
- lifting actions although no stroke is installed, etc.

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer
2. The mobile robot shall report an error of type ‘INVALID_ORDER_ACTION’ with level ‘WARNING’ and the erroneous fields as errorReferences
3. The warning shall be reported until the mobile robot has accepted a new order.

6.1.4.4 Mobile robot receives an order with the same orderId, but a lower orderUpdateId than the current orderUpdateId

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer.
2. The mobile robot shall keep the previous order in its buffer.
3. The mobile robot shall report an error of type ‘OUTDATED_ORDER_UPDATE’ and level ‘WARNING’.
4. The mobile robot shall continue with executing the previous order.
5. The warning shall be reported until the mobile robot has accepted a new order.

6.1.4.5 Mobile robot receives an order with the same orderId and same orderUpdateId as the current orderUpdateId

Example:

- Fleet control resends the order because it did not yet receive any state message with the respective `orderUpdateId`.

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer.
2. The mobile robot shall keep the previous order in its buffer.
3. Reporting depends on the content of the message:
   - If the content of the new order is the same as the content of the previous one, the mobile robot shall ignore the new order.
   - If the content of the new order differs, the mobile robot shall report an error of type ‘SAME_ORDER_UPDATE_ID’ and level ‘WARNING’.
4. The mobile robot shall continue with executing the previous order.
5. The warning shall be reported until the mobile robot has accepted a new order.

6.1.4.6 Mobile robot receives an order with orderId different to the orderId of an active order

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer.
2. The mobile robot keeps the previous order in its buffer.
3. The mobile robot shall report an error of type ‘OTHER_ORDER_ACTIVE’ and level ‘WARNING’.
4. The mobile robot shall continue with executing the previous order.
5. The warning shall be reported until the mobile robot has accepted a new order.

6.1.4.7 Mobile robot receives an order with the start node being out of range

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer.
2. The mobile robot shall report an error of type ‘START_NODE_OUT_OF_RANGE’ and level ‘WARNING’.
3. The warning shall be reported until the mobile robot has accepted a new order.

6.1.4.8 Mobile robot receives an order with at least one node not being reachable

Resolution:

4. The mobile robot shall not take over the new order in its internal buffer.
5. The mobile robot shall report an error of type ‘NO_ROUTE_TO_TARGET’ and level ‘WARNING’.
6. The warning shall be reported until the mobile robot has accepted a new order.

6.1.4.9 Mobile robot receives an order while in an operating mode that does not allow new orders

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer.
2. The mobile robot shall report an error of type ‘MOBILE_ROBOT_NOT_AVAILABLE’ and level ‘WARNING’.
3. The warning shall be reported until the mobile robot is in an order mode that allows for new orders.

6.1.4.10 Mobile robot receives an order containing nodes with unknown mapId

Resolution:

1. The mobile robot shall not take over the new order in its internal buffer.
2. The mobile robot shall report an error of type ‘UNKNOWN_MAP_ID’ and level ‘WARNING’.
3. The warning shall be reported until the mobile robot has accepted a new order.

6.1.5 Corridors

The optional `corridor` edge attribute allows the mobile robot to deviate from the edge trajectory for obstacle avoidance and defines the boundaries within which the mobile robot is allowed to operate. To use the `corridor` attribute, a predefined trajectory is required that the mobile robot would follow if no `corridor` attribute was defined. This can be either the trajectory defined on the mobile robot known to the fleet control or the trajectory sent in an order. The behavior of a mobile robot using the `corridor` attribute is still the behavior of a line-guided mobile robot, except that it is allowed to temporarily deviate from a trajectory to avoid obstacles. Note that a corridor communicated within an order is released for the mobile robot by default. If the `releaseRequired` flag is set to true, the mobile robot shall request approval from fleet control before using the corridor as described in chapter 6.6.10 Request use of Corridors.

Remark:
An edge inside an order defines a logical connection between two nodes and not necessarily the (real) trajectory that a mobile robot follows when driving from the start node to the end node. Depending on the mobile robot type, the trajectory that a mobile robot takes between the start and end nodes is either defined by fleet control via the trajectory edge attribute or assigned to the mobile robot as a predefined trajectory. Depending on the internal state of the mobile robot, the selected trajectory may vary.

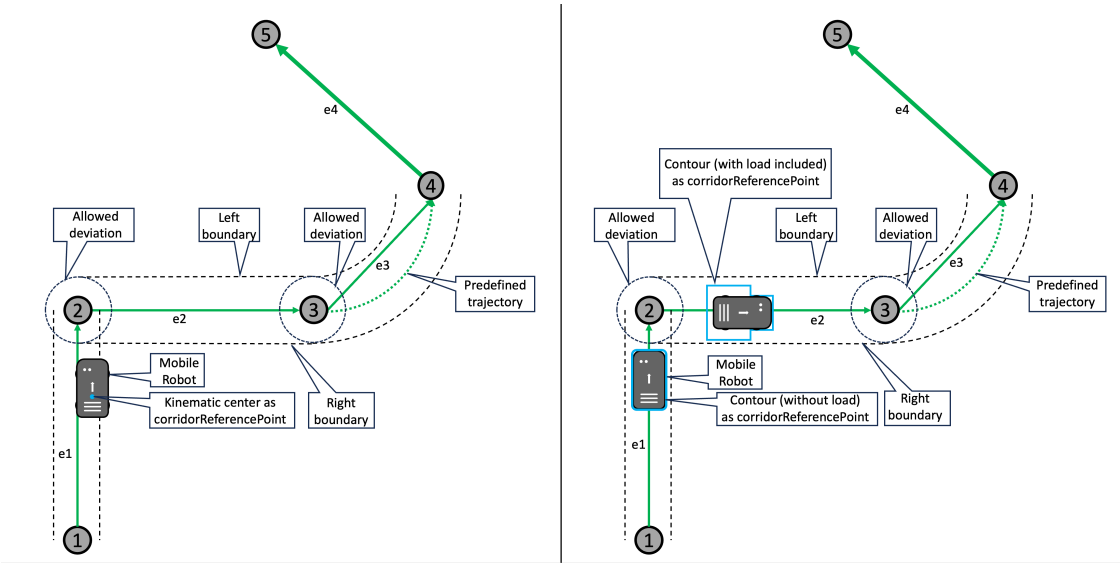


Figure 10: Edges with a corridor attribute that defines the left and right boundaries within which a mobile robot is allowed to deviate from its predefined trajectory to avoid obstacles

On the left, the kinematic center defines the allowed deviation, while on the right, the contour of the mobile robot, possibly extended by the load, defines the allowed deviation. This is defined by the `corridorReferencePoint` parameter. The area in which the mobile robot is allowed to navigate independently (and deviate from the original edge trajectory) is defined by a left and a right boundary. The optional `corridorReferencePoint` field specifies whether the mobile robot control point or the mobile robot contour should be inside the defined boundary. The boundaries of the edges shall be defined in such a way that the mobile robot is inside the boundaries of the new and now current edge as soon as it passes a node. Instead of setting the corridor boundaries to zero, fleet control shall not use the `corridor` attribute if the mobile robot shall not deviate from the trajectory.

The mobile robot's motion control software shall constantly check that the mobile robot is within the defined boundaries. If not, the mobile robot shall stop because it is out of the allowed navigation space and report an error of type 'OUTSIDE_OF_CORRIDOR' with level 'CRITICAL'. The fleet control can decide if user interaction is required or if the mobile robot can continue by canceling the current order and sending a new order to the mobile robot with corridor information that allows the mobile robot to move again.

Remark:
Allowing the mobile robot to deviate from the trajectory increases the possible footprint of the mobile robot during driving. This circumstance shall be considered during initial operation, and when the fleet control makes a traffic control decision based on the mobile robot's footprint. See also Section 6.6.2 Traversal of nodes and edges for further information.

6.2 Actions

If the mobile robot supports actions other than driving, these actions are instructed via the `actions` array that is attached to a node or an edge, sent via the separate topic `instantActions` (see section 6.2.1 Instant actions) or configured via action zones (see section 6.4.1 Zone types). Actions that are to be executed on an edge shall only run while the mobile robot is on the edge (see Section 6.6.2 Traversal of nodes and entering/leaving edges, triggering of actions).

Actions that are triggered on nodes can run as long as they need to run and should be self-terminating (e.g., an audio signal that lasts for five seconds or a pick action, that is finished after picking up a load) or formulated pairwise (e.g., "activateWarningLights" and "deactivateWarningLights").

6.2.1 Instant Actions

In certain cases, it is necessary to send actions to the mobile robot that need to be performed immediately. This is possible by publishing an `instantAction` message to the topic `instantActions`. These actions shall not conflict with the content of the mobile robot's current order (e.g., `instantAction` to lower fork, while order says to raise fork).

Some examples for which instant actions could be relevant are: - pause the mobile robot without changing anything in the current order - resume order after pause - activate signal (optical, audio, etc.)

When a mobile robot receives an `instantAction`, an appropriate `actionStatus` shall be added to the `instantActionStates` array of the mobile robot's state. The `actionStatus` shall be updated according to the progress of the action. See also Figure 11 for the different transitions of an `actionStatus`. The `blockingType` of an instant action is always 'NONE'.

When the mobile robot receives an `instantAction` it cannot execute, it shall report an 'INVALID_INSTANT_ACTION' error with level 'WARNING' and the `actionId` of the `instantAction` as `errorReference`.

6.2.2 Action blocking types and sequence

The order of multiple actions in a list defines the sequence in which the mobile robot shall execute them.

The parallel execution of actions is governed by their respective `blockingType`. Actions can have four distinct blocking types, described in Table 3.

-	Parallel execution allowed	Parallel execution not allowed
Automatic driving allowed	NONE	SINGLE
Automatic driving not allowed	SOFT	HARD

Table 3: Definition of action blocking types dependent on driving and parallel execution

Figure 11 describes how the mobile robot shall handle the blocking type of actions. Whenever the mobile robot arrives at a point where new actions are to be executed (i.e., when it reaches a node, edge, or action zone), the actions are enqueued in the same sequence as the actions array. This queue is continually processed as shown in Figure 11. If the blocking type of any action in the queue is 'SOFT' or 'HARD', the mobile robot shall stop automatic driving. Actions are collected for parallel execution if the action's blocking type is 'NONE' or 'SOFT'. If an action with blocking type 'SINGLE' or 'HARD' is to be executed, all collected parallel actions shall be 'FINISHED' or 'FAILED' before starting the action. If there are no more actions with blocking type 'SOFT' or 'HARD' in the queue, the mobile robot can resume automatic driving. 'FINISHED' or 'FAILED' actions shall be removed from the queue.

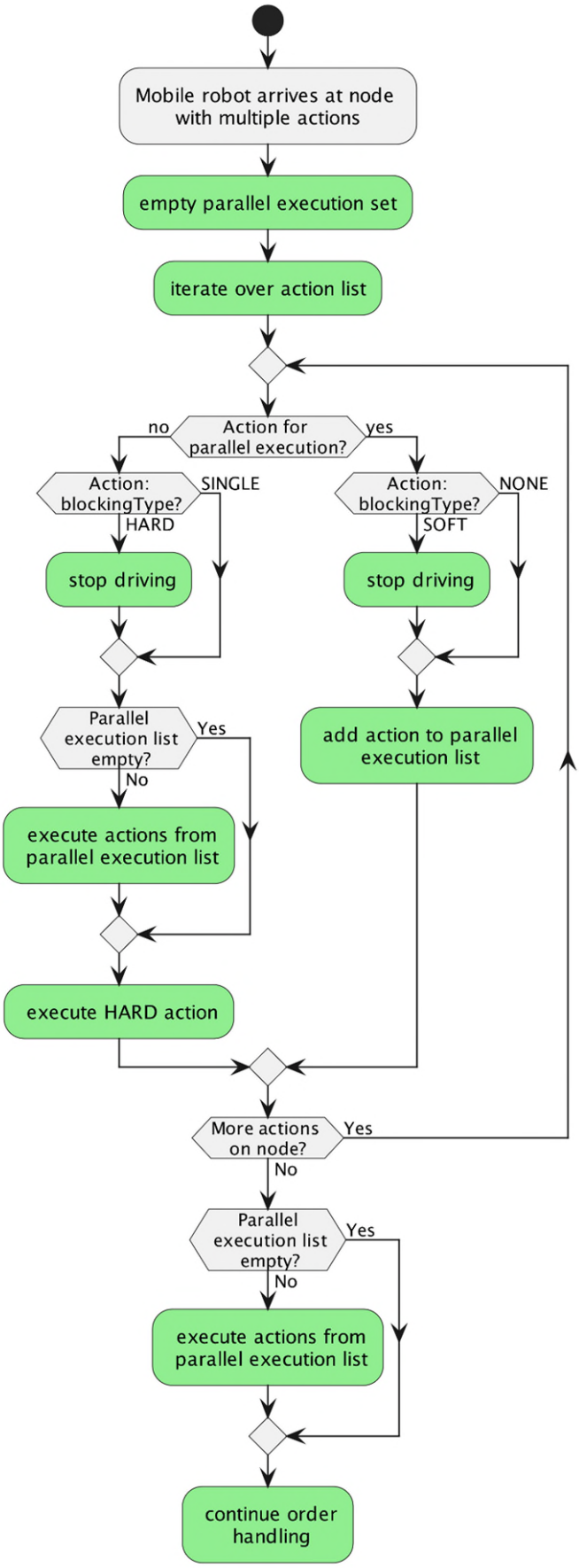


Figure 11: Handling multiple actions

6.2.3 Predefined Actions

This section presents predefined actions that shall be used by the mobile robot, if the mobile robot's capabilities map to the action description. If there is a sensible way to use the defined parameters, they shall be used. Additional parameters can be defined, if they are needed to execute an action successfully. The actions `cancelOrder`, `startPause` and `stopPause` shall be supported by every mobile robot.

If there is no way to map some action to one of the actions of the following section, the mobile robot manufacturer can define additional actions that shall be used by fleet control.

6.2.3.1 Definition, parameters, effects and scope

action type	counter action	description	idem-potent	parameters	linked state	instant	node	edge	zone
startPause	stopPause	Activates the pause mode. A linked state is required, because many mobile robots can be paused by using a hardware switch. No more automatic driving - reaching next node is not necessary. Actions that can be paused (`pauseAllowed=true`), shall be paused, other actions continue. Order execution is resumed after stopPause.	yes	-	paused	yes	no	no	no
stopPause	startPause	Deactivates the pause mode. Movement and all other actions will be resumed (if any).A linked state is required because many mobile robots can be paused by using a hardware switch. stopPause can also restart mobile robots that were stopped with a hardware button that triggered startPause (if configured).	yes	-	paused	yes	no	no	no
startHibernation	stopHibernation	Initiates hibernate mode, in which the mobile robot shall remain connected to the MQTT broker but no longer needs to send state messages. The mobile robot shall report this action as ‘FINISHED’ before discontinuing publishing state messages and publish a connection state of ‘HIBERNATING’. If the mobile robot has an active order, it shall clear it. Reaching the next node is not required.While in ‘HIBERNATING’ connection state, mobile robot shall not be moving. The mobile robot shall only receive and respond to the instant action ‘stopHibernation’ and shall not respond to any other commands, such as orders or additional instant actions. If the mobile robot’s battery becomes critically low while in this mode, the mobile robot may stop ‘HIBERNATING’ autonomously to report an error. In case a wake-up time is set, the	yes	wakeUpTime (string, optional)	-	yes	no	no	

action type	counter action	description	idem-potent	parameters	linked state	instant	node	edge	zone
		mobile robot is able to autonomously exit the ‘HIBERNATING’ connection state at the specified time and will publish the corresponding connection state transition before resuming normal operation.							
stopHibernation	startHibernation	Ends hibernate mode. To initiate wake-up while the mobile robot is in the ‘HIBERNATING’ state, a control device (onboard or external) shall subscribe to the `instantAction` topic and remain connected to the MQTT broker. Because the mobile robots standard control device may be partially shut down during hibernation, the wake-up may be triggered by a distinct MQTT client (separate from the mobile robots usual communication client). Upon success, the mobile robot shall publish the connection state ONLINE.	yes	-	-	yes	no	no	
shutdown	-	Initiates a coordinated shutdown of the mobile robot, where it disconnects from the MQTT broker. The execution of the shutdown action requires the mobile robot to be in an idle state. There is no way using the VDA 5050 protocol to automatically restart due to the connection being terminated.If a mobile robot is in hibernate mode but should be shut down, it shall first exit hibernation (via stopHibernation) before executing shutdown.	yes	-	-	yes	no	no	no
startCharging	stopCharging	Activates the charging process. Charging can be done on a charging spot (mobile robot stopped) or on a charging lane (while driving). Protection against overcharging is the responsibility of the mobile robot.	yes	-	powerSupply.charging	yes	yes	no	no

action type	counter action	description	idem-potent	parameters	linked state	instant	node	edge	zone
stopCharging	startCharging	Discontinues the charging process. The charging process can also be interrupted by the mobile robot or the charging station, e.g., if the battery is full.	yes	-	powerSupply.charging	yes	yes	no	no
initializePosition	-	Resets (overrides) the pose of the mobile robot with the given parameters.	yes	x (float64)y (float64)theta (float64)mapId (string)lastNodeId (string)	mobileRobotPosition.xmobileRobotPosition.ymobileRobotPosition.thetamobileRobotPosition.mapIdlastNodeId maps	yes	yes(Elevator)	no	no
enableMap	-	Enable a previously downloaded map explicitly to be used in orders without initializing a new position.	yes	mapId (string)mapVersion (string)	maps	yes	yes	no	no
downloadMap	-	Trigger the download of a new map. Active during the download. Errors reported in mobile robot state. Finished after verifying the successful download, preparing the map for use and setting the map in the state.	yes	mapId (string)mapVersion (string)mapDownloadLink (string)mapHash (string, optional)	maps	yes	no	no	no
deleteMap	-	Trigger the removal of a map from the mobile robot’s memory.	yes	mapId (string)mapVersion (string)	maps	yes	no	no	no
downloadZoneSet	-	Trigger the download of a zone set. Active during the download. Errors reported in mobile robot state. Finished after verifying the successful download, preparing the zone set for use and setting the zone set in the state.	yes	zoneSetId (string)zoneSetDownloadLink (string)zoneSetHash (string, optional)	zoneSets	yes	no	no	no
enableZoneSet	-	Enable a previously downloaded zone set explicitly to be used in orders.	yes	zoneSetId (string)	zoneSets	yes	yes	no	no
deleteZoneSet	-	Trigger the removal of a zone set from the mobile robot’s memory.	yes	zoneSetId (string)	zoneSets	yes	no	no	no

action type	counter action	description	idem-potent	parameters	linked state	instant	node	edge	zone
clearInstantActions	-	Removes all finished or failed instant actions from the mobile robot state.	yes	-	instantActionStates	yes	yes	no	no
clearZoneActions	-	Removes all finished or failed zone actions from the mobile robot's state.	yes	-	zoneActionStates	yes	yes	no	no
stateRequest	-	Requests the mobile robot to send a new state message.	yes	-	-	yes	no	no	no
logReport	-	Requests the mobile robot to generate and store a log report.	yes	reason(string)	-	yes	no	no	no
pick	drop (if automated)	Request the mobile robot to pick a load. Mobile robots with multiple load handling devices can process multiple pick operations in parallel. In this case, the parameter lhd needs to be present (e.g., LHD1). The parameter stationType informs how the pick operation is handled in detail (e.g., floor location, rack location, passive conveyor, active conveyor, etc.). The load type informs about the load unit and can be used to switch field for example (e.g., EPAL, INDU, etc). For preparing the load handling device (e.g., pre-lift operations based on the height parameter), the action could be announced in the horizon in advance. But, pre-Lift operations, etc., are not reported as 'RUNNING' in the mobile robot state, because the associated node is not released yet.If on an edge, the mobile robot can use its sensing device to detect the position for picking the node.	no	lhd (string, optional) stationType (string, optional) stationName (string, optional) loadType (string, optional) loadId (string, optional) height (float64, optional) defines bottom of the load related to the floordepth (float64, optional) for forkliftsside (string, optional) e.g., conveyor	.load	no	yes	yes	no

action type	counter action	description	idem-potent	parameters	linked state	instant	node	edge	zone
drop	pick (if automated)	Request the mobile robot to drop a load. See action pick for more details.	no	lhd (string, optional) stationType (string, optional) stationName (string, optional) loadType (string, optional) loadId (string, optional) height (float64, optional) depth (float64, optional) …	.load	no	yes	yes	no
detectObject	-	Mobile robot detects object(e.g., load, charging spot, free parking position).	yes	objectType (string, optional)	-	no	yes	yes	yes
finePositioning	-	On a node, mobile robot will position exactly on a target.The mobile robot is allowed to deviate from its node position.On an edge, the mobile robot will e.g., align on stationary equipment while traversing an edge.	yes	stationType (string, optional) stationName (string, optional)	-	no	yes	yes	yes
waitForTrigger	-	Mobile robot shall wait for a trigger of the type defined specified in the triggerType parameter, which is an array of strings. Two predefined values shall be used when semantically appropriate: ‘FLEET_CONTROL’ if the trigger originates from the fleet control, and ‘LOCAL’ if the trigger comes from an input on the mobile robot (e.g., button press, manual loading). If none of the predefined values meet the specific requirements, custom values can be defined. Fleet control is responsible for handling the timeout and shall cancel the order if necessary.	yes	triggerType [string] (array)	-	no	yes	no	yes

action type	counter action	description	idem-potent	parameters	linked state	instant	node	edge	zone
trigger	-	Fleet control system notifies the mobile robot that a waitForTrigger action has been released. Typically, this occurs when the fleet control system receives information from a third-party system indicating that the process the mobile robot was waiting for has completed.	yes	-	-	yes	no	no	no
retry	-	Mobile robot retries action defined via actionId that is currently in state RETRIABLE.	yes	actionId (string)	-	yes	no	no	no
skipRetry	-	Mobile robot shall skip the action defined via actionId that is currently in state RETRIABLE, setting action to FAILED.	yes	actionId (string)	-	yes	no	no	no
cancelOrder	-	Mobile robot stops as soon as possible. This could be immediately or on the next node. See Chapter 6.1.3 Order cancellation.	yes	orderId (string, optional)	-	yes	no	no	no
factsheetRequest	-	Requests the mobile robot to send a factsheet	yes	-	-	yes	no	no	no
updateCertificate	-	Request the mobile robot to download and activate a new certificate set, the service parameter is an extensible enum with the predefined parameter ‘MQTT’ to be used for mqtt connection.	yes	service (string) keyDownloadLink (string) certificateDownloadLink (string) certificateAuthorityDownloadLink (string, optional)	-	yes	no	no	no

Table 4: Predefined actions and their scope (instant, node, edge, zone)

6.2.3.2 Action states

action type	'INITIALIZING'	'RUNNING'	'PAUSED'	'FINISHED'	'FAILED'	'RETRIABLE'
startPause	-	Activation of the mode is in preparation.If the mobile robot supports an instant transition, this state can be omitted.	-	Mobile robot is not moving. All pauseable actions are paused. The pause mode has been activated. The mobile robot reports paused: "true".	The pause mode cannot be activated for some reason (e.g., overridden by hardware switch).	
stopPause	-	Deactivation of the mode is in preparation. If the mobile robot supports an instant transition, this state can be omitted.	-	The pause mode has been deactivated. All paused actions are resumed. The mobile robot reports paused: "false".	The pause mode cannot be deactivated for some reason (e.g., overridden by hardware switch).	-
startHibernation	-	Activation of the hibernate mode is in preparation. If the mobile robot supports an instant transition, this state can be omitted.	-	Mobile robot is not moving. The active order has been cleared, if any. No state messages are sent by the mobile robot.Hibernate mode has been activated. The mobile robot reports connection state "HIBERNATING".	The HIBERNATING connection state could not be published (e.g., overridden by a hardware switch).	-
stopHibernation	-	Deactivation of the hibernate mode is in preparation. If the mobile robot supports an instant transition, this state can be omitted.	-	Hibernate mode has been deactivated.The mobile robot reports connectionState "ONLINE".	The hibernate mode could not be deactivated (e.g., overridden by a hardware switch).	-

action type	'INITIALIZING'	'RUNNING'	'PAUSED'	'FINISHED'	'FAILED'	'RETRIABLE'
shutdown	-	Activation of the OFFLINE connection state is in preparation. If the mobile robot supports an instant transition, this state can be omitted.	-	Mobile robot is not moving. The connection between mobile robot and broker is terminated in a coordinated way. The mobile robot reports connection state "OFFLINE".	The shutdown cannot be executed for some reason (e.g., mobile robot is not in idle state, overridden by a hardware switch).	-
startCharging	-	Activation of the charging process is in progress (communication with charger is running). If the mobile robot supports an instant transition, this state can be omitted.	-	The charging process has been started. The mobile robot reports powerSupply.charging: "true".	The charging process could not be started for some reason(e.g., not aligned to charger). Charging problems should correspond with an error.	The charging process could not be initiated. The mobile robot is waiting for intervention from fleet control or an operator.
stopCharging	-	Deactivation of the charging process is in progress (communication with charger is running). If the mobile robot supports an instant transition, this state can be omitted.	-	The charging process has been stopped. The mobile robot reports powerSupply.charging: "false"	The charging process could not be stopped for some reason (e.g., not aligned to charger). Charging problems should correspond with an error.	-
initializePosition	-	Initializing of the new pose in progress (confidence checks, etc.). If the mobile robot supports an instant transition, this state can be omitted.	-	The pose has been reset. The mobile robot reports mobileRobotPosition.x = x, mobileRobotPosition.y = y, mobileRobotPosition.theta = theta mobileRobotPosition.mapId = mapId mobileRobotPosition.lastNodeId = lastNodeId	The pose is not valid or cannot be reset. General localization problems should correspond with an error.	-

action type	'INITIALIZING'	'RUNNING'	'PAUSED'	'FINISHED'	'FAILED'	'RETRIABLE'
downloadMap	Initialize the connection to the map server.	Mobile robot is downloading the map.	-	The download has finished. Mobile robot updates its state by setting the mapId/mapVersion and the corresponding mapStatus to 'DISABLED'.	The download failed, updated in mobile robot state (e.g., connection lost, Map server unreachable, mapId/mapVersion not existing on map server).	Download failed or was interrupted. The mobile robot is waiting for intervention from fleet control.
enableMap	-	The mobile robot enables the map with the requested mapId and mapVersion and disables any other map with the same mapId.	-	The map has been enabled. The mobile robot updates the corresponding mapStatus of the requested map to 'ENABLED' and the other versions with same mapId to 'DISABLED'.	The requested combination of mapId/mapVersion does not exist.	-
deleteMap	-	Mobile robot deletes map with requested mapId and mapVersion from its internal memory.	-	The map has been deleted. The mobile robot removes mapId/mapVersion from its state.	The map could not be deleted, e.g., because map is currently in use or requested combination of mapId/mapVersion has already been deleted before.	-
downloadZoneSet	Initialize the connection to the zone set server.	Mobile robot is downloading the zone set.	-	The download has finished. The mobile robot updates its state by setting a corresponding zoneSet object in its state with zoneSetStatus 'DISABLED'.	The download failed, updated in mobile robot state (e.g., connection lost, server unreachable, zone set not existing, zone set with same zoneSetId already on mobile robot).	Download failed or was interrupted. The mobile robot is waiting for intervention from fleet control.

action type	'INITIALIZING'	'RUNNING'	'PAUSED'	'FINISHED'	'FAILED'	'RETRIABLE'
enableZoneSet	-	Mobile robot enables the zone set with the requested zoneSetId and disables any other zone set for the same mapId.	-	The zone set has been enabled. The mobile robot updates the corresponding zoneSetStatus of the requested zoneSet to 'ENABLED' and the other zone sets for the same mapId to 'DISABLED'.	The requested zone set does not exist.	-
deleteZoneSet	-	Mobile robot deletes the zone set with requested zoneSetId from its internal memory.	-	The zone set has been deleted. The mobile robot removes zoneSet object from its state.	The zone set could not be deleted, deleted, e.g., because zone set is currently in use or the requested zone set has already been deleted before.	-
clearInstantActions	-		-	The instant actions array has been cleaned from all FINISHED or FAILED instantActions.	-	-
clearZoneActions	-		-	The zone actions array has been cleaned from all FINISHED or FAILED instantActions.	-	-
stateRequest	-	-	-	The state has been communicated	-	-
logReport	-	The report is being generated. If the mobile robot supports an instant generation, this state can be omitted.	-	The report has been stored. The name of the log is reported as part of the action state.	The report can not be stored (e.g., no space).	-

action type	'INITIALIZING'	'RUNNING'	'PAUSED'	'FINISHED'	'FAILED'	'RETRIABLE'
pick	Initializing of the pick process, e.g., outstanding lift operations.	The pick process is running (mobile robot is moving into station, load handling device is busy, communication with station is running, etc.).	The pick process is being paused, e.g., if a safety field is violated. After removing the violation, the pick process continues.	Pick has been done. Load has entered the mobile robot and mobile robot reports new load state.	Pick failed, e.g., station is unexpected empty. Failed pick operations should correspond with an error.	Pick failed, but is retriable. The mobile robot is waiting for intervention from fleet control or an operator.
drop	Initializing of the drop process, e.g., outstanding lift operations.	The drop process is running (mobile robot is moving into station, load handling device is busy, communication with station is running, etc.).	The drop process is being paused, e.g., if a safety field is violated. After removing the violation the drop process continues.	Drop has been done. Load has left the mobile robot and mobile robot reports new load state.	Drop failed, e.g., station is unexpected occupied. Failed drop operations should correspond with an error.	Drop failed, but is retriable. The mobile robot is waiting for intervention from fleet control or an operator.
detectObject	-	Object detection is running.	-	Object has been detected.	Could not detect the object.	Object detection failed, but is retriable. The mobile robot is waiting for intervention from fleet control or an operator.
finePositioning	-	Mobile robot positions itself exactly on a target.	The fine positioning process is being paused,e.g., if a safety field is violated. The fine positioning continues after e.g. the violation had been resolved.	Goal position in reference to the station has been reached.	Goal position in reference to the station could not be reached.	Fine positioning failed but is retriable. The mobile robot is waiting for intervention from fleet control or an operator.
waitForTrigger	-	Mobile robot is waiting for the trigger	-	Trigger has been triggered.	waitForTrigger fails, if order has been canceled.	-

action type	‘INITIALIZING’	‘RUNNING’	‘PAUSED’	‘FINISHED’	‘FAILED’	‘RETRIABLE’
cancelOrder	-	Mobile robot is stopping or driving, until it reaches the next node.	-	Mobile robot is not moving. Mobile robot has canceled executing the order and is in idle state.	Mobile robot has no active order. The previous order has already been canceled. Passed orderId does not match the currently active orderId.	-
factsheetRequest	-	-	-	The factsheet has been communicated	-	-
updateCertificate	-	Mobile robot is downloading and installing certificates	-	Certificates have been downloaded, installed and are active.	Download or installation failed.	-

Table 5: Expected behavior in action states of predefined actions

6.2.3.3 Update mobile robot certificate

For security reasons, mobile robot communication (at least for fleet management) should be secured. Typically, communication to the MQTT broker is secured via TLS, which requires one or more root certificates and a mobile robot-specific key pair.

The parameter `service` specifies the service (e.g., 'MQTT') for which the certificates are to be used. The parameter `certificateAuthorityDownloadLink` specifies the URL for the root certificate(s). The parameters `certificateDownloadLink` and `keyDownloadLink` specify the URLs for the mobile robot-specific public and private keys.

The download shall be secured via TLS as well, since the sender of the instantAction cannot be verified. It is also advisable to validate the certificate chain before it is activated.

6.3 Maps

To ensure consistent navigation among different types of mobile robots, the position is always specified in reference to the project-specific coordinate system (see Figure 12). The project-specific coordinate system is referring to the coordinate system that is defined for the interaction between fleet control and the mobile robot. For the differentiation between different levels of a site or location, a unique `mapId` is used. The map coordinate system is to be specified as a right-handed coordinate system with the z-axis pointing skywards. A positive rotation therefore is to be understood as a counterclockwise rotation. The mobile robot coordinate system is also specified as a right-handed coordinate system (ISO 9787 4.1) with the x-axis pointing in the forward direction of the mobile robot and the z-axis pointing upward (ISO 9787 5.5). The mobile robot reference point is defined as (0,0,0) in the mobile robot reference frame, unless specified otherwise.

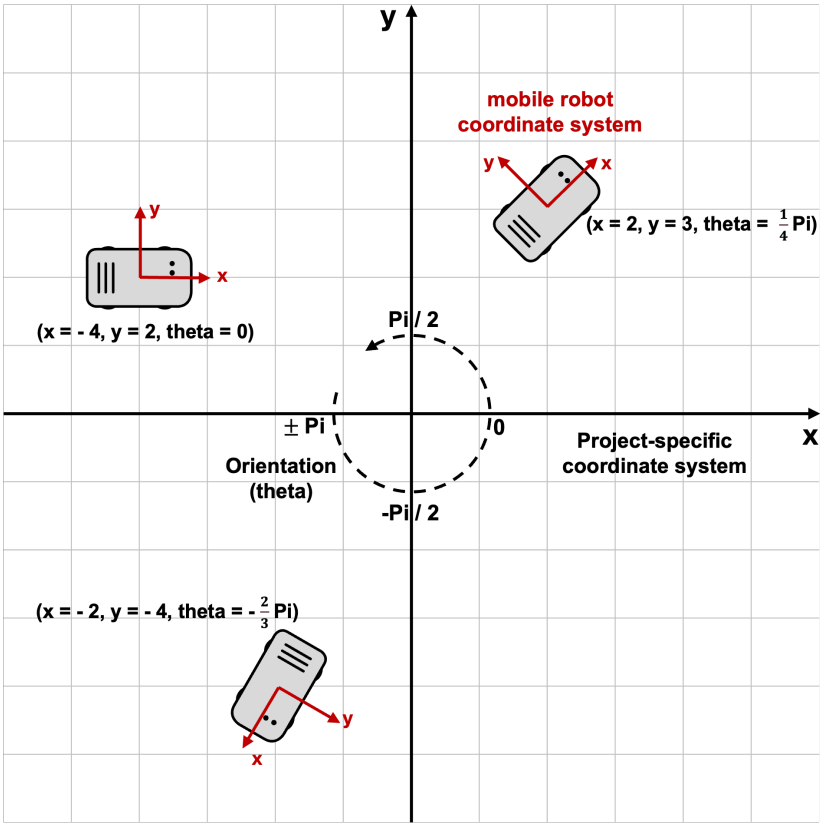


Figure 12: Coordinate system with sample mobile robot and orientation

The X, Y, and Z coordinates shall be given in meters. The orientation shall be in radians and shall be within -Pi and +Pi.

6.3.1 Map distribution

To enable an automatic map distribution and intelligent management of restarting the mobile robots if necessary, fleet control can manage the maps on the mobile robot.

The map files to be distributed are stored on a dedicated map server that is accessible by the mobile robots. To ensure efficient transmission, each transmission should consist of a single file. If multiple maps or files are required, they should be bundled or packed into a single file. The process of transferring a map from the map server to a mobile robot is a pull operation, initiated by the fleet control triggering a download command using an `instantAction`.

Each map is uniquely identified by a combination of a map identifier (field `mapId`) and a map version (field `mapVersion`). The map identifier describes a specific area of the mobile robot's physical workspace, and the map version indicates updates to previous versions. Before accepting a new order, the mobile robot shall check that there is a map on the mobile robot for each map identifier in the requested order. If a corresponding `mapId` is missing in the list of available maps, the mobile robot shall report an error of type 'UNKNOWN_MAP_ID' and level 'WARNING'. It is the responsibility of the fleet control to ensure that the correct maps are enabled to operate the mobile robot.

In order to minimize downtime and make it easier for the fleet control to synchronize the process of enabling of new maps, maps shall be pre-loaded or buffered on the mobile robots. The status of the maps on the mobile robot is reflected in the mobile robot's state. Transferring a map to a mobile robot and enabling the map are different processes. To enable a pre-loaded map on a mobile robot, the fleet control shall send an instant action. As a result, any other map with the same map identifier but a different map version shall be disabled by the mobile robot.

Deletion of maps can also be done by the fleet control via an instant action.

The map distribution process is shown in Figure 13.

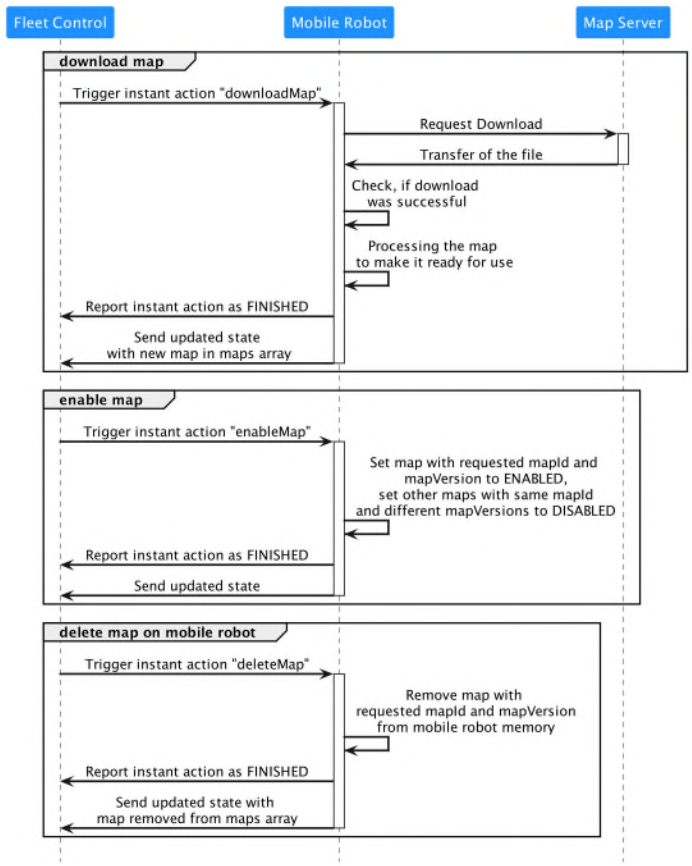


Figure 13: Communication required between fleet control, mobile robot and map server to download, enable, and delete a map

6.3.2 Maps in the mobile robot state

The `mapId` field in the `mobileRobotPosition` of the state represents the currently active map.

Information about the maps available on a mobile robot is presented in the `maps` array, which is a component of the state message. Each entry in this array is a JSON object consisting of the mandatory fields `mapId`, `mapVersion`, and `mapStatus`, which can be either ‘ENABLED’ or ‘DISABLED’. An ‘ENABLED’ map can be used by the mobile robot if necessary. A ‘DISABLED’ map shall not be used. The status of the download process is indicated by the current action not being completed. Errors are also reported in the state. Note that multiple maps with different `mapId` can be enabled at the same time. There shall only be one version of maps with the same `mapId` enabled at a time. If the `maps` array is empty, no maps are currently available on the mobile robot.

6.3.3 Map download

The map download shall be triggered by the `downloadMap` instant action from the fleet control. It shall contain the mandatory parameters `mapId` and `mapDownloadLink` under which the map is stored on the map server and which can be accessed by the mobile robot.

The mobile robot sets the `actionStatus` to ‘RUNNING’ as soon as it starts downloading the map file. If the download is successful, the `actionStatus` is updated to ‘FINISHED’. If the download is unsuccessful, the status is set to ‘FAILED’. Once the download has been successfully completed, the map shall be added to the array of `maps` in the state. Maps shall not be reported in the state until they are ready to be enabled.

The process of downloading a map shall not modify, delete, enable, or disable any existing maps on the mobile robot. The mobile robot shall reject the download of a map with a `mapId` and `mapVersion` that is already on the mobile robot. An error of type ‘DUPLICATE_MAP’ and level ‘WARNING’ shall be reported, and the status of the instant action shall be set to ‘FAILED’. The fleet control shall first delete the map on the mobile robot and then restart the download.

6.3.4 Enable downloaded maps

There are two ways to enable a map on a mobile robot:

1. **Fleet control enables map**: Use the `enableMap` instant action to set a map to ‘ENABLED’ on the mobile robot. Other Versions of the same `mapId` with different `mapVersion` are set to ‘DISABLED’.
2. **Manually enable a map on the mobile robot**: In some cases, it might be necessary to enable the maps on the mobile robot directly. The result shall be reported in the mobile robot state.

Fleet control shall ensure that the correct maps are activated on the mobile robot when sending the corresponding `mapId` as part of a `nodePosition` in an order. If the mobile robot is to be set to a specific position on a new map, the `initializePosition` instant action shall be used.

6.3.5 Delete maps on the mobile robot

The fleet control can request the deletion of a specific map from a mobile robot. This shall be done by using the instant action `deleteMap`. When a mobile robot runs out of memory, it should report this to the fleet control, which can then initiate the deletion of maps. The mobile robot itself shall not delete maps. After successfully deleting a map, the mobile robot shall remove the corresponding entry from its `maps` array in the state message.

6.4 Zones

Zones are used to define rules for specific areas of the mobile robot workspace. In this way, zones allow mobile robots to navigate freely between nodes while giving the fleet control the ability to manage traffic. Zones can be used to locally deny mobile robots access to areas or to link access to conditions (zone types: ‘BLOCKED’ and ‘RELEASE’). It is also possible to enforce specific behavior while within the zone (zone types: ‘LINE_GUIDED’, ‘SPEED_LIMIT’, ‘COORDINATED_REPLANNING’, and ‘ACTION’) or influence the driving behavior by incentivizing or penalizing certain areas (zone types: ‘PRIORITY’ and ‘PENALTY’) or giving a predefined driving direction (zone types: ‘DIRECTED’, ‘BIDIRECTED’). The zone types are defined in the following sections.

Potential conflicts in orders due to overlapping of zones or combination of zone and edge properties and how to resolve them are addressed in section 6.4.4 Interaction between zones. For released nodes that are part of the order but are restricted due to zones (e.g., node located within a ‘BLOCKED’ or ‘RELEASE’ zone), the robot is expected to act according to the zones (e.g., not enter or wait for ‘GRANTED’ state of the request). Some mobile robots cannot process zones at all, while other mobile robots might only be able to work with a certain subset of zone types, such as ‘BLOCKED’. All mobile robots shall therefore report to fleet control which zones they are able to understand by adding the according zone names to the `supportedZones` array under `typeSpecifications` in their factsheet. Also (virtually) line-guided mobile robots can choose to support zone-based navigation if they can implement the logic of the corresponding zone types defined in the following. A zone set shall only be changed and distributed by fleet control to keep consistency in the system.

6.4.1 Zone types

Two categories of zones are distinguished: contour-based zones and kinematic center-based zones. This distinction is based on the different conditions for when the mobile robot is considered to be entering and exiting zones.

6.4.1.1 Contour-based zones

For contour-based zones, the contour of the mobile robot (including its load) determines zone entry and exit. Any part of the contour entering the zone is a zone entry. As soon as no part of the mobile robot’s contour remains within the zone, it is a zone exit.

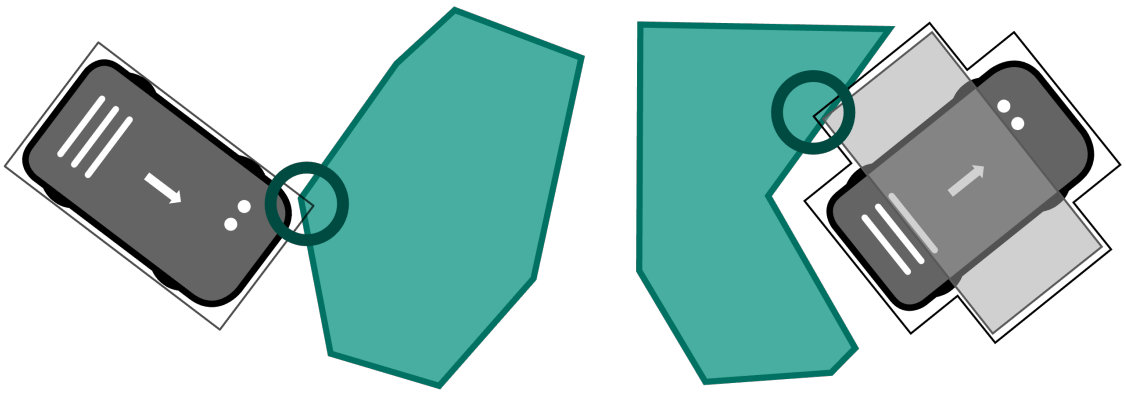


Figure 14: Depiction of a mobile robot entering a zone based on its contour (left) and a loaded mobile robot with corresponding extended bounding box exiting a zone (right)

The following contour-based zones are defined:

Zone Type	Zone Parameters	Data type	Description
BLOCKED	none		Mobile robots shall not enter this zone. If a mobile robot has entered the zone or finds itself within one, it shall stop and throw an 'BLOCKED_ZONE_VIOLATION' error with level set to 'CRITICAL'.
LINE_GUIDED	none		No free navigation is allowed in this zone, mobile robots shall follow the predefined trajectories on edges. Mobile robots may only enter this zone if the route is explicitly specified by the fleet control in the form of a node-edge graph. Any movement of the mobile robot that requires it to enter this zone shall follow a predefined trajectory. When entering the zone, the mobile robot shall be on the trajectory of the edge that crosses the zone. The edges that enter and are inside the line-guided zone require a trajectory sent from the fleet control or a predefined trajectory on the mobile robot. A corridor can be sent to allow the mobile robot to deviate from the trajectory.
RELEASE		-	Mobile robots are only allowed entering this zone once they have been granted access through fleet control.
	releaseLossBehavior	string	Enum {'STOP', 'CONTINUE', 'EVACUATE'} When the access to this zone is revoked or expired, the mobile robot can either 'STOP', 'CONTINUE', or 'EVACUATE' the zone. This action is only executed, when the mobile robot is already in the zone and the release expires or is revoked. If not defined, the mobile robot is expected to STOP and report an error.'STOP': Mobile robot stops and sends a 'RELEASE_LOST' error with level 'CRITICAL'.'EVACUATE': Execute the evacuation behavior of the mobile robot to leave the zone, keeping the `zoneRequest` object granting release in its state until the zone is left.'CONTINUE': If the release is revoked or expires after the mobile robot has already entered the zone, the mobile robot continues its path, keeping the `zoneRequest` object granting the zone release in its state. If the order ends inside the zone, the mobile robot waits for a new order.
COORDINATED_REPLANNING	none		No autonomous replanning is allowed within this zone. Mobile robots are only allowed adjusting their path if granted permission by fleet control.
SPEED_LIMIT			Mobile robots shall not drive faster than the defined maximum speed within this zone.
	maximumSpeed	float64	Maximum permitted speed for mobile robot within the zone in m/s. The speed limit shall already be reached upon entering the zone.
ACTION			The mobile robot shall perform predefined actions when entering, traversing, or exiting the zone. The factsheet defines which actions can be executed when.

Zone Type	Zone Parameters	Data type	Description
	entryActions[action]	array	Actions to be triggered when entering the zone. Empty array, if no actions required.
	duringActions[action]	array	Actions to be executed while crossing the zone. Empty array, if no actions required.
	exitActions[action]	array	Actions to be triggered when leaving the zone. Empty array, if no actions required.

Table 6: Contour-based zone types and their parameters

6.4.1.2 Kinematic center-based zones

In kinematic center-based zones, the mobile robot's kinematic center determines its entry and exit of the zones. When the mobile robot's kinematic center is inside a zone, the mobile robot shall follow the defined behavior. 'PRIORITY' and 'PENALTY' zones are zones which only influence the path planning of mobile robots. 'DIRECTED' zones define a preferred direction of travel within the zone. 'BIDIRECTED' zones define a travel direction and its opposite direction to be used. Other directions shall be avoided. The `directedLimitation` and `bidirectedLimitation` enums specify the limits within which the mobile robot may deviate from its direction of travel. The direction of travel is the velocity vector in the project-specific coordinate system.

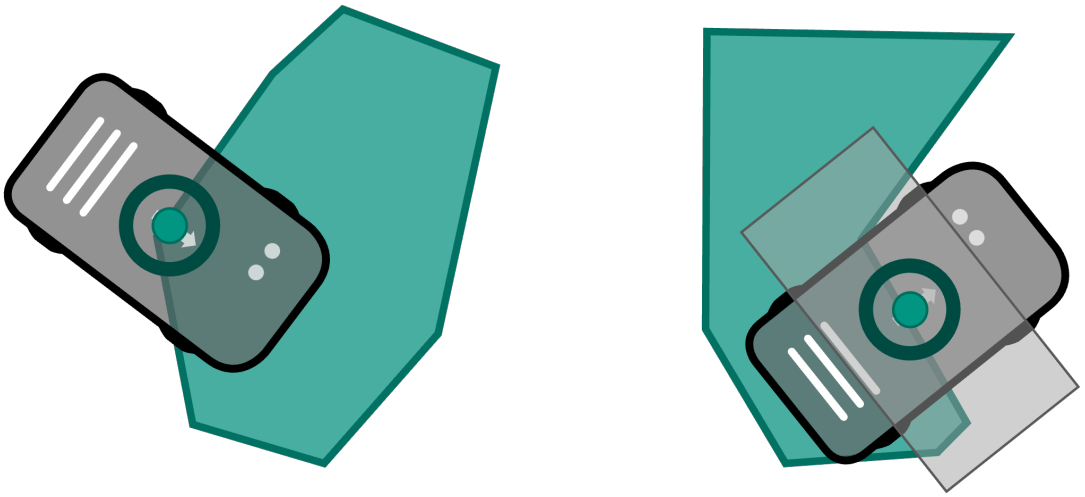


Figure 15: Depiction of a mobile robot entering a zone based on its kinematic center (left) and a loaded mobile robot exiting a zone based on its kinematic center (right)

Zone Type	Zone Parameters	Data type	Description
PRIORITY			The workspace encompassed by this zone is associated with an incentive for the mobile robot to plan its route through this zone compared to an otherwise equivalent area without such a zone on the map.
	priorityFactor	float64	[0.0…1.0] Relative factor that determines the preference of the zone over a workspace without a zone. 0.0 means no preference, as if there was no zone, 1.0 is maximum preference.
PENALTY			The workspace encompassed by this zone is associated with a disincentive for the mobile robot to plan its route through this zone compared to an otherwise equivalent area without such a zone on the map.

Zone Type	Zone Parameters	Data type	Description
	penaltyFactor	float64	[0.0…1.0] Relative factor that determines the penalty of the zone compared to a workspace without that zone. 0.0 means no penalty, as if there was no zone, 1.0 is the maximum penalty, causing the mobile robot to take this path only if it cannot find any other feasible route.
DIRECTED			Mobile robots shall traverse this zone in a specific direction of travel.
	direction	float64	Preferred direction of travel within the zone in radians. The direction of travel is the angular orientation of the mobile robot's velocity vector in the project-specific coordinate system.
	directedLimitation	string	Enum {'SOFT','RESTRICTED','STRICT'} SOFT: Mobile robots may deviate from the defined direction of travel, but should avoid it, RESTRICTED: The mobile robot may deviate from the defined direction of travel, e.g., to avoid an obstacle, but shall never traverse opposite to the defined direction of travel, STRICT: The mobile robot shall maintain the defined direction of travel as precisely as its technical capabilities allow.
BIDIRECTED			While in this zone, mobile robots shall only move in the defined direction of travel and its direct opposite (+ Pi), mobile robots should not cross this zone in any other direction.
	direction	float64	Preferred direction of travel within the zone in radians. The direction of travel is the angular orientation of the mobile robot's velocity vector in the project-specific coordinate system.
	bidirectedLimitation	string	Enum {'SOFT', 'RESTRICTED'} <>SOFT: Mobile robots may deviate from the defined directions of travel, but should avoid it, RESTRICTED: The mobile robot shall not traverse in any other direction than the directions of travel, except for obstacle avoidance.

Table 7: Kinematic center-based zone types and their parameters

6.4.2 Zone set transfer

Zone sets shall only be changed and distributed by fleet control to keep consistency in the system. The preferred way to distribute zone sets is via the `zoneSet` topic. If the mobile robot supports zones, the update via the `zoneSet` topic shall be supported. Larger zone sets can also be shared through the `downloadZoneSet` instant action, following the map distribution concept in figure 13.

A `zoneSet` is an array of `zone` objects with a globally unique identifier, `zoneSetId`. It is associated with a single map referenced through the `mapId`. The `mapVersion` shall not be referenced, as the same zone set might be intended to be used for several versions of one map. In general, several zone sets can be defined in addition to a single map and it is upon fleet control to ensure that the right zone set is enabled for each map on the mobile robot. As with maps, the `zoneSetStatus` indicates which zone set is currently used by the mobile robot. Only a single zone set can be active at once for each `mapId` on the mobile robot. Zones shall not extend beyond the spatial boundaries of a map. The content of a zone set with a unique `zoneSetId` shall not change. If changes are required within a zone set, it shall be referenced with a new `zoneSetId`.

The `zoneSetStatus` of a newly added zone set shall always be set to ‘DISABLED’ and shall be enabled through the `enableZoneSet` instant action before use.

If the mobile robot receives a new zone set via the `zoneSet` topic or `downloadZoneSet` instant action with the same `zoneSetId` as an existing one, it shall not take over the zone set in its internal memory and report an error of type ‘DUPLICATE_ZONE_SET’ and level ‘WARNING’ for a reasonable amount of time for the fleet control to notice that the zone update failed.

6.4.3 Communication for interactive zones

For communicating requests for the interactive zones ‘RELEASE’ and ‘COORDINATED_REPLANNING’, the field `zoneRequests` in the state message is used. The separate topic `responses` is used by fleet control to respond to these requests.

Before entering an interactive zone, the mobile robot shall state a request. A request before entry of an interactive zone is necessary, even if the order contains released nodes within the zone. The mobile robot decides at which point before entering the zone to make its requests. If the response is not received in time, the mobile robot shall not enter the zone.

Requests shall only be made for zones of enabled zone sets. Zone requests can also be made for zone sets belonging to maps that the mobile robot is not currently on.

The `requestId` allows fleet control to distinguish between different requests and allows the mobile robot to issue several alternative requests for the same zone at the same time. Each request attempt shall use a unique identifier per mobile robot. Ids can be reused after a mobile robot restart.

For requests to enter a ‘RELEASE’ zone, a `zoneRequest` object of `requestType` ‘ACCESS’ shall be added to the state message. For permission to enter a ‘COORINATED_REPLANNING’ zone with a planned path or for replanning its path within the zone, the `requestType` shall be set to ‘REPLANNING’. For a ‘REPLANNING’ request, the planned path shall be added as NURBS to the `trajectory` field of the `zoneRequest`. Multiple requests with different trajectories for the same zone can be made. Each path shall be requested with its own `zoneRequest` object. If a mobile robot requires access to a workspace covered by two or more ‘RELEASE’ zones, it shall request access and receive approval for all necessary zones before entering the area. If a mobile robot navigates through a workspace on the map that is covered by two or more ‘COORDINATED REPLANNING’ zones, it shall request its path within this area individually for each zone and receive approval from the fleet control before entering or changing paths.

The parameter `requestStatus` shall be initially set to ‘REQUESTED’ by the mobile robot when stating its request.

Fleet control responds to zone requests via the `responses` topic. The response message contains an array of `response` objects. Each `response` shall only respond to a single request referenced by the `requestId`. Each response has a `responseType` that is either ‘GRANTED’, ‘QUEUED’, ‘REVOKED’, or ‘REJECTED’. If the `responseType` is ‘GRANTED’, the mobile robot is allowed to enter the zone or use the requested trajectory. Fleet control can set the `responseType` to ‘QUEUED’ to acknowledge the mobile robot’s request without giving permission, informing the mobile robot that its request is being processed. If the `responseType` is ‘REJECTED’, the mobile robot shall not enter the zone or use the requested trajectory. The `responseType` ‘REVOKED’ indicates that the permission is no longer valid. The fleet control shall assume a ‘REVOKED’ request as still being ‘GRANTED’, until the `requestStatus` of the mobile robot is set to ‘REVOKED’. The `response` object can include a `leaseExpiry` which specifies until when a ‘GRANTED’ request is valid. To extend the `leaseExpiry` fleet control can resend a response message with an updated `leaseExpiry` time.

The mobile robot shall acknowledge the fleet controls response by setting the `requestStatus` accordingly and keep the request for as long as it considers the information relevant. See also Section 6.9 Request/response mechanism.

The interaction between the mobile robot and the fleet control for ‘RELEASE’ zones shall be according to Figure 16.

While the mobile robot remains in the ‘RELEASE’ zone, it keeps the `zoneRequest` object in its state and continues to report `requestStatus` as ‘GRANTED’ to inform fleet control that it is still inside the zone. After mobile robot has exited the zone, it shall remove the corresponding `zoneRequest` entry from its state message. When receiving a response with `responseType` ‘REVOKED’, the mobile robot shall remove the request from its state. When the `leaseExpiry` has passed, the requestStatus shall be set to ‘EXPIRED’ and the zone shall not be entered. If the mobile robot is already inside the ‘RELEASE’ zone when the `leaseExpiry` has passed or the request is ‘REVOKED’, it shall report a warning and react according to the `releaseLossBehavior` defined in the zone definition.

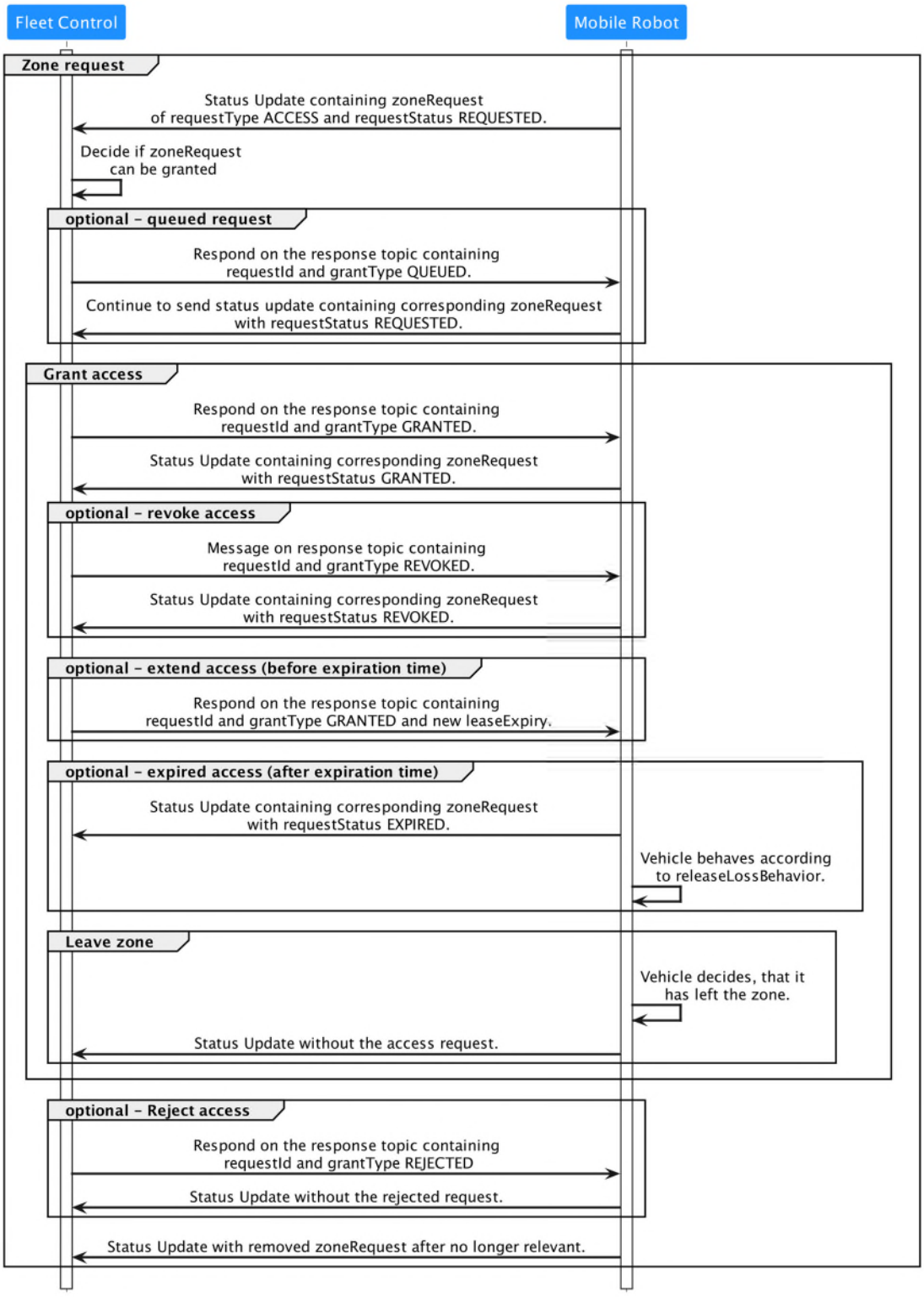


Figure 16: Zone request behavior for a RELEASE zone

The interaction between the mobile robot and the fleet control for ‘COORDINATED_REPLANNING’ zones shall be according to Figure 17.

The mobile robot shall choose one of the trajectories of all ‘GRANTED’ requests to the zone and set the corresponding `requestStatus`to ‘GRANTED’ while removing all other requests from its state. When receiving a response with `responseType` ‘REVOKED’, the mobile robot shall remove the request from its state and not enter the ‘COORDINATED_REPLANNING’ zone. When the `leaseExpiry` has passed, the `requestStatus` shall be set to ‘EXPIRED’

and the zone shall not be entered. If the mobile robot is already inside the ‘RELEASE’ zone when the `leaseExpiry` has passed or the request is ‘REVOKED’, it shall stop driving and report a warning. To continue, the mobile robot shall state a new request.

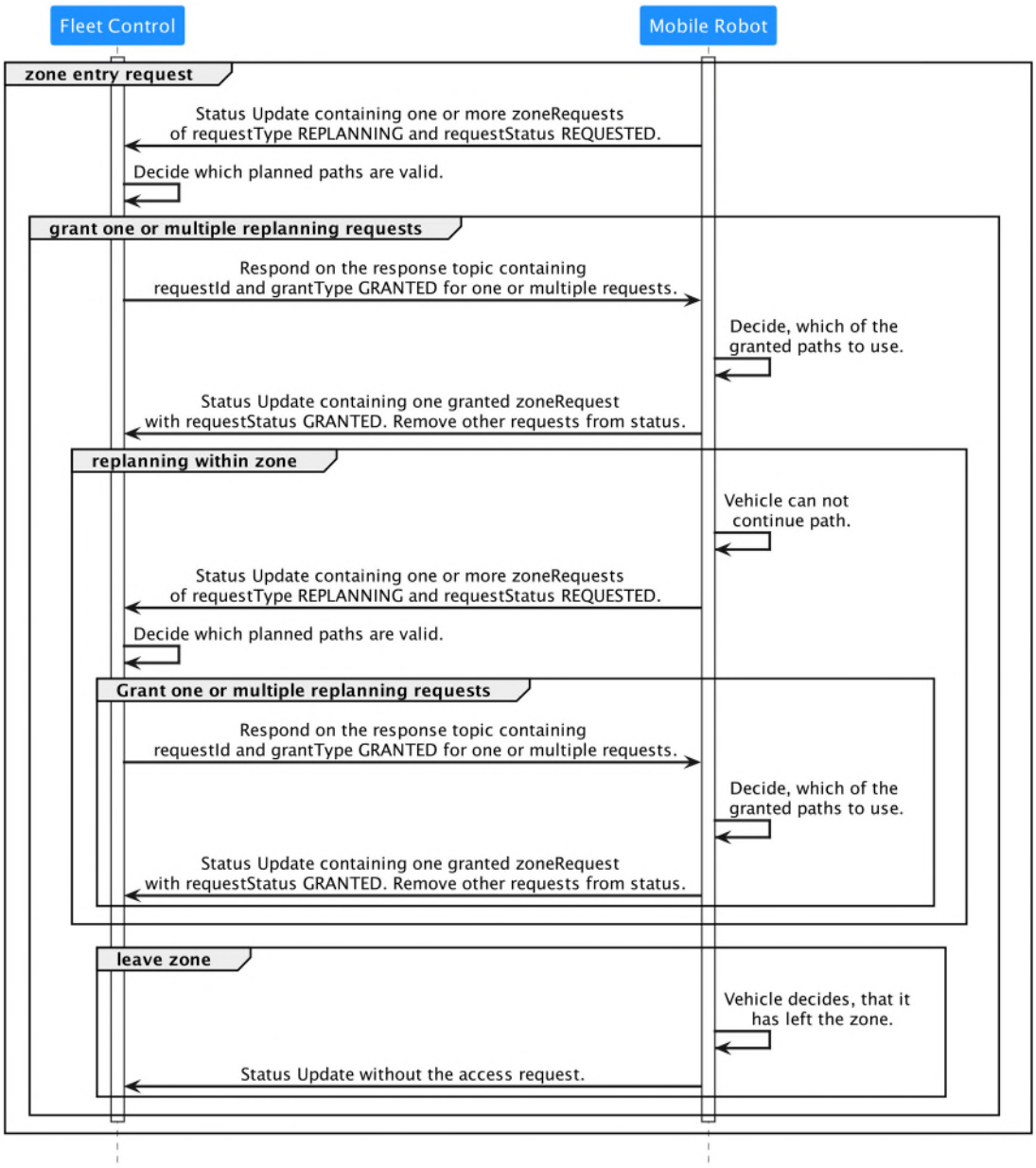


Figure 17: Zone request behavior for a COORDINATED_REPLANNING zone

6.4.4 Interactions between zones

In the following matrix possible interactions between zones are described. The matrix is symmetric, as the interaction between two zones is the same, regardless of the order in which they are considered. For each combination, there is either a zone behavior that is overrulling the other (e.g., a ‘BLOCKED’ zone overrules a ‘LINE_GUIDED’ zone) or there is no conflict (e.g., a ‘LINE_GUIDED’ zone and a ‘COORDINATED_REPLANNING’ zone). ‘DIRECTED’ and ‘BIDIRECTED’ zones shall not overlap, since this might lead to an undefined behavior. The column *No Zone* defines the behavior for contour-based zones, where mobile robots can be inside a defined zone type and an area without a zone at the same time. For kinematic center-based zones the mobile robot can only be completely within or outside the zone, so there is no possible interaction.

	BLOCKED	RELEASE	LINE_GUIDED	COORDINATED_REPLANNING	SPEED_LIMIT	ACTION	PRIORITY	PENALTY	DIRECTED	BIDIRECTED	No Zone	EDGE-PROPER-TIES
BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED	BLOCKED
RELEASE		No Conflict	No Conflict	No Conflict	No Conflict	No Conflict	No Conflict	No Conflict	No Conflict	No Conflict	No Conflict	No Conflict
LINE_GUIDED			No conflict	LINE_GUIDED	No Conflict	(1)	LINE_GUIDED	LINE_GUIDED	LINE_GUIDED	No conflict	LINE_GUIDED	No conflict
COORDINATED_REPLANNING				(2)	No conflict	(1)	No conflict	No conflict	No conflict	No conflict	COORDINATED_REPLANNING	(3)
SPEED_LIMIT					(4)	No conflict	No conflict	No conflict	No conflict	No conflict	SPEED_LIMIT	(4)
ACTION						(5)	No conflict	No conflict	No conflict	No conflict	ACTION	(5)
PRIORITY							(6)	(6)	No conflict	No conflict	(7)	No conflict
PENALTY								(6)	No conflict	No conflict	(7)	No conflict
DIRECTED									(8)	(8)	(7)	(9)
BIDIRECTED										(8)	(7)	(9)

Table 8: Interaction matrix for zones

1) If actions would conflict with other zones' behavior, report a 'ZONE_ACTION_CONFLICT' error with level 'CRITICAL' (order error) and stop the mobile robot.
2) Planned trajectory required to be granted for all 'COORDINATED_REPLANNING' zones.
3) If a trajectory is predefined for the edge, it shall be sent in the zone request.
4) The lowest of the competing `maximumSpeed` values applies.
5) Execute all actions.
6) The most restrictive one is always selected here; for PRIORITY zones, the lowest `priorityFactor` is used; for overlapping PRIORITY and PENALTY zones, the highest `penaltyFactor` is used; for overlapping PENALTY zones, the highest `penaltyFactor` is used.
7) For kinematic center-based zones the mobile robot can only be completely within or outside the zone, so this overlap is not possible.
8) Zones shall not overlap, since the behavior is not defined.
9) A `trajectory` as part of the edge properties shall override the directed and bidirected zones.

6.4.5 Error handling within zones

If at any point of the order execution, a mobile robot realizes, that it can not reach a node in its order, it shall report a ‘NODE_UNREACHABLE’ error with level ‘CRITICAL’ to the fleet control. The fleet control shall then decide how to proceed. The mobile robot shall not try to reach the node again, but wait for further instructions from the fleet control.

6.5 Connection

During the connection of a mobile robot client to the broker, a last will topic and message shall be set, which is published by the broker upon disconnection of the mobile robot client from the broker. Thus, the fleet control can detect a disconnection event by subscribing the connection topics of all mobile robots. The disconnection is detected via a heartbeat that is exchanged between the broker and the client. Thus, the fleet control can detect a disconnection event by subscribing to the `connection` topic of each mobile robot.

As a result, the timestamp and headerId fields will always be outdated.

Mobile robot wants to disconnect gracefully:

1. Mobile robot sends “vda5050/v3/manufacturer/serialNumber/connection” with `connectionState` set to `OFFLINE`.
2. Disconnect the MQTT connection with a disconnect command.

Mobile robot comes online:

1. Set the last will to “vda5050/v3/manufacturer/serialNumber/connection” with the field `connectionState` set to ‘CONNECTION_BROKEN’, when the MQTT connection is created.
2. Send the topic “vda5050/v3/manufacturer/serialNumber/connection” with `connectionState` set to ‘ONLINE’.

All messages on this topic shall be sent with a `retained` flag.

When connection between the mobile robot and the broker stops unexpectedly, the broker will send the last will to the topic: “vda5050/v3/manufacturer/serialNumber/connection” with the field `connectionState` set to ‘CONNECTION_BROKEN’.

6.6 State

The mobile robot state shall be published on a single topic. Compared to separate messages (e.g., for current order progress, battery state and errors), using a single topic reduces the workload of both the broker and the fleet control system when handling messages, while also keeping the mobile robot state information synchronized.

The mobile robot state message shall be published when relevant events occur or at least every 30 seconds.

The following events shall trigger a transmission of the state message: - Receiving an order - Receiving an order update - Changes in the `load` object - Change in the `errors` array - Change in the `operatingMode` field - Change in the `driving` field - Change in the `paused` field - Change in the `safetyState` object - Change in the `newBaseRequest` field - Change in the `lastNodeId` or `lastNodeSequenceId` field - Change in the `edgeRequests` or `zoneRequests` arrays - Change in the `powerSupply.charging` field - Change in the `nodeStates` or `edgeStates` arrays - Change in the `actionStates`, `instantActionStates` or `zoneActionStates` arrays - Change in the `zoneSets` array - Change in the `maps` array

Remark:
For above mentioned arrays, changes in the individual items of the array as well as adding or removing entries shall trigger a state message transmission.

There should be an effort to curb the amount of communication. If two events correlate with each other (e.g., the receiving of a new order usually forces an update of the `nodeStates` and `edgeStates`; as does the driving over a node), it is sensible to trigger one state update instead of multiple. The minimum time between two consecutive state messages is defined by the factsheet (7.10 Implementation of the factsheet message `protocolLimits.timing.minimumStateInterval`).

6.6.1 Concept and logic

The order progress is tracked by the `nodeStates` and `edgeStates`. Additionally, if the mobile robot is capable of determining its current position, it shall publish it via the `mobileRobotPosition` field.

The `nodeStates` and `edgeStates` include all upcoming nodes and edges for the mobile robot to traverse.

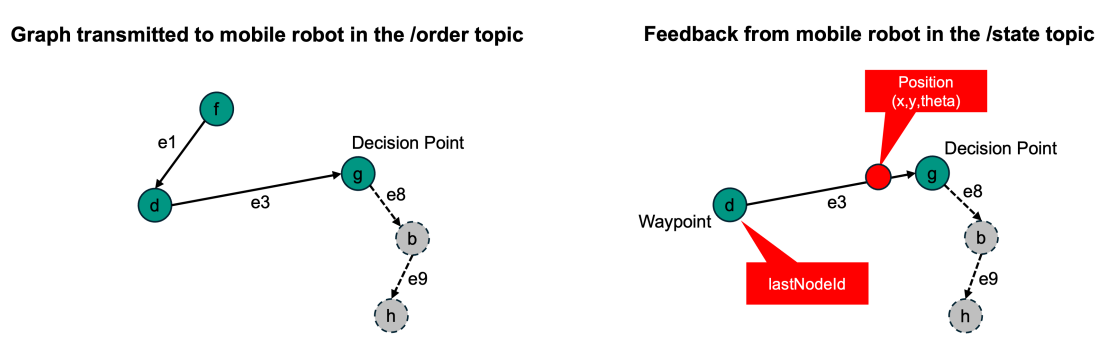


Figure 18: Order information provided by the state topic. Only the ID of the last node and the remaining nodes and edges are transmitted

6.6.2 Traversal of nodes and edges

The mobile robot decides on its own when a node should count as traversed. A requirement for the traversal is that the mobile robot's control point shall be within the node's `allowedDeviationXY` and its orientation within `allowedDeviationTheta`. The `allowedDeviationXY` defines at what point a line-guided mobile robot can deviate from its predefined trajectory, to cut the corner along a smoother path rather than reaching the node's exact position. When leaving the `allowedDeviationXY` the mobile robot shall be back on its predefined trajectory of the subsequent edge. If the edge attribute `corridor` of the subsequent edge is set, these boundaries should be met additionally.

In case the mobile robot is located too far away from the first node of an order, the fleet control can add an extended `allowedDeviationXY` to this node to include the mobile robot's current position.

The mobile robot shall report the traversal of a node by removing its `nodeState` from the `nodeStates` array and setting the `lastNodeId` and `lastNodeSequenceId` to the traversed node's values.

As soon as the mobile robot reports the node as traversed, the mobile robot shall trigger the actions associated with the node, if any. The traversal of a node also necessarily implies leaving the edge that is leading up to the node. The edge shall then also be removed from the `edgeStates` and the actions that were active on the edge shall be finished.

The traversal of the node also marks the moment when the mobile robot enters the following edge, if there is one. The edge's actions shall be triggered, if any. An exception to this rule is if the mobile robot shall stop on the node (because of a soft or hard blocking action) – then the mobile robot only enters the following edge once it begins driving again.

When an active order exists, the fields `lastNodeId` and `lastNodeSequenceId` shall be updated only when the mobile robot traverses a released node that is part of this order. For

example if a physically line-guided mobile robot detects a physical marker/tag that is not part of the active order's `nodes`, this detection shall not lead to a change of `lastNodeId` or `lastNodeSequenceId`.

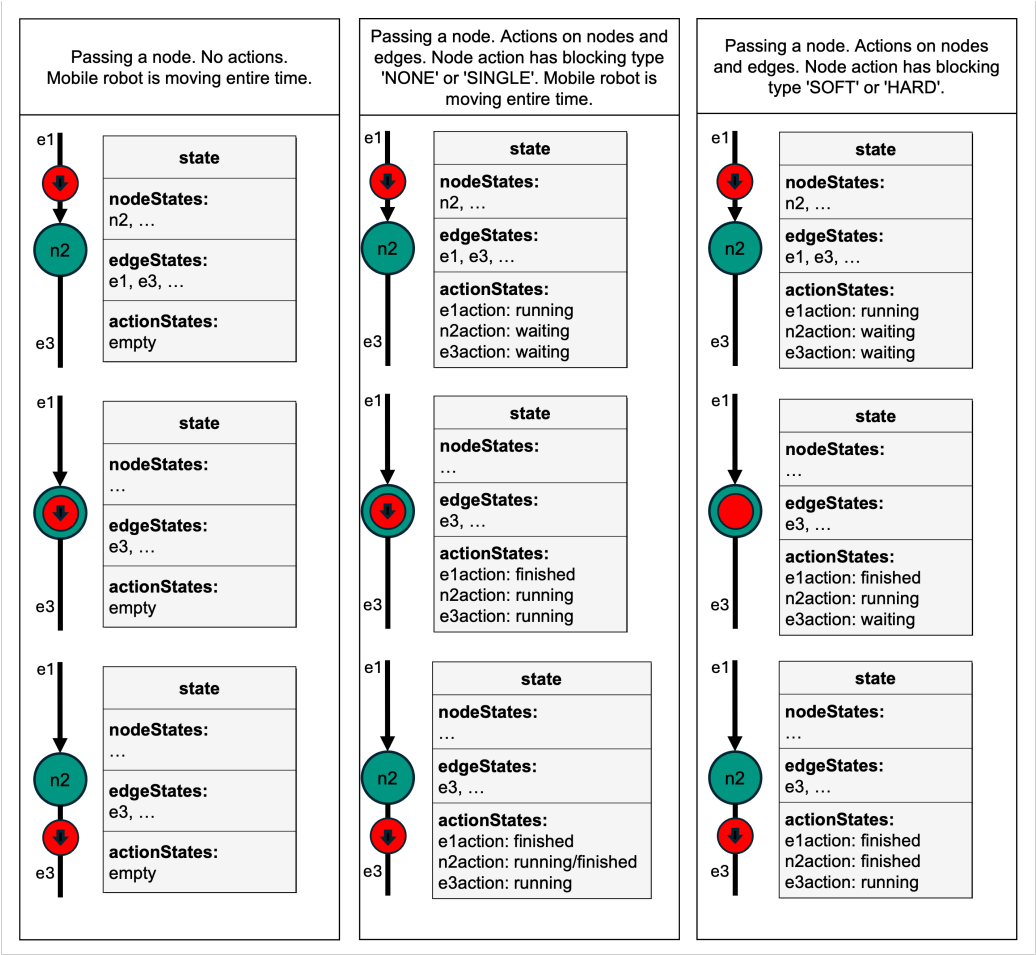


Figure 19: Depiction of nodeStates, edgeStates, and actionStates during order handling

6.6.2.1 Definition of allowedDeviationXY as an ellipse

The allowedDeviationXY is defined as an ellipse around the node position to allow more flexible approaches to the node.

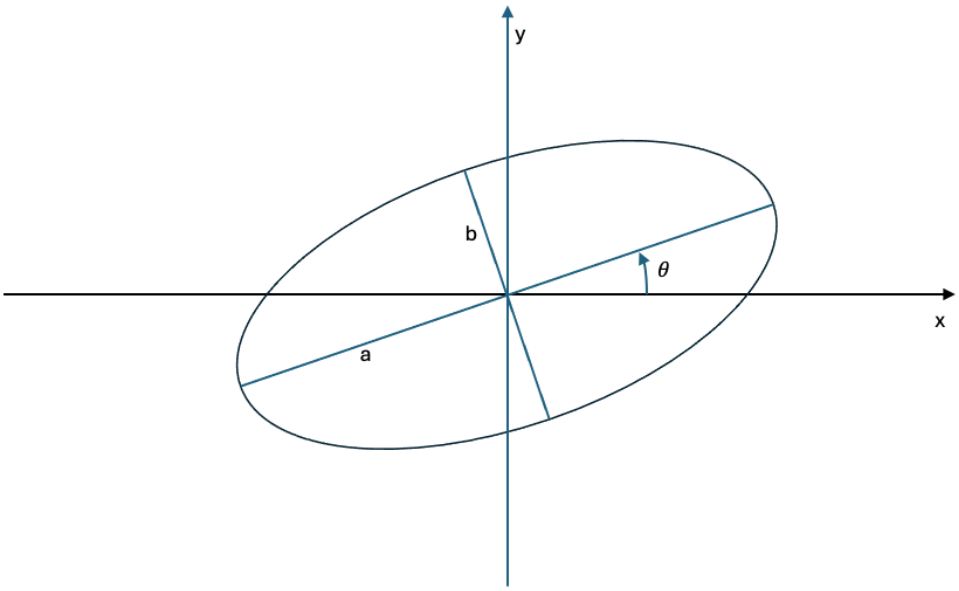


Figure 20: allowedDeviation ellipse

6.6.3 Base request

If the mobile robot detects that its base is running short, it can set the `newBaseRequest` flag to “true” to attempt to prevent unnecessary braking.

6.6.4 Information

The mobile robot can submit arbitrary additional information to the fleet control via the `information` array. It is up to the mobile robot to decide how long it reports information via an information message.

The fleet control shall not use the information for logic; they shall only be used for visualization and debugging purposes.

6.6.5 Errors

The mobile robot reports any issues via the `errors` array.

6.6.5.1 Error levels

The issues can have four levels: ‘WARNING’, ‘URGENT’, ‘CRITICAL’, and ‘FATAL’.

- A ‘WARNING’ level issue does not require immediate attention. The mobile robot can continue its current order and is able to take new orders. The error might be self-resolving, e.g., a dirty LiDar-scanner.
- An ‘URGENT’ level issue, e.g., a low battery level, requires immediate attention. The mobile robot can continue its current order and is able to take new orders.
- A ‘CRITICAL’ level issue requires immediate attention, e.g., trying to pick an object, that is not there. The mobile robot shall not continue driving since it can not continue its current order but is able to take new orders.
- A ‘FATAL’ level issue requires user intervention, e.g., losing localization. The mobile robot shall not continue driving since it can neither continue its currently active order nor take any new orders.

The mobile robot can add references that help with finding the cause of the error via the `errorReferences` array. The fields `errorDescription` and `errorHint` may provide human-readable text explaining the error or suggesting a possible resolution.

Regardless of the level of the issue, the mobile robot shall never clear its order due to it.

6.6.5.2 Error references

If an error occurs due to an erroneous order or execution failure, the mobile robot can return meaningful error references in the field `errorReferences` to support finding the cause of the error. This can include the following information:

- `headerId`
- Topic (`order` or `instantAction`)
- `orderId` and `orderUpdateId` if error was caused by an order update
- `actionId` if error was caused by an action
- List of parameters if error was caused by erroneous action parameters

6.6.5.3 Error translations

For both `errorDescription` and `errorHint`, the mobile robot can provide translations by using the `errorDescriptionTranslations` and `errorHintTranslations` arrays. Each translation consists of an ISO 639-1 language code and the corresponding translated text.

6.6.5.4 Predefined error types

The mobile robot shall use predefined error types to report specific issues. The following table lists the predefined error types and their description.

Error Type	Error level	Description	Reference	Report duration
‘UNSUPPORTED_PARAMETER’	‘CRITICAL’	Receival of message with an unsupported optional parameter.	Name of parameter	Until new order is accepted.
‘NO_ORDER_TO_CANCEL’	‘WARNING’	The mobile robot received a `cancelOrder` action, but it does not have an active order to cancel.	`actionId` of `cancelOrder`	Until new order is accepted.
‘VALIDATION_FAILURE’	‘WARNING’	Receival of malformed order.	If possible, `orderId` and `orderUpdateId` of rejected message.	Until new order is accepted.
‘INVALID_ORDER_ACTION’	‘WARNING’	Receival of an order containing unsupported actions.	`orderId` and `orderUpdateId` of rejected message.	Until new order is accepted.
‘INVALID_INSTANT_ACTION’	‘WARNING’	Receival of an unsupported instant action.	`actionId` of `instantAction`	Until new instant action is accepted.
‘OUTDATED_ORDER_UPDATE’	‘WARNING’	Receival of an order with correct `orderId` but outdated `orderUpdateId`.	`orderId` and `orderUpdateId` of rejected message.	Until new order is accepted.
‘SAME_ORDER_UPDATE_ID’	‘WARNING’	Receival of a duplicate order message (same `orderId` and `orderUpdateId`)	`orderId` and `orderUpdateId` of rejected message.	Until new order is accepted.
‘ORDER_UPDATE_FOLLOWING_CANCEL’	‘WARNING’	Receival of an order update for an order that has already been cancelled.	`orderId` and `orderUpdateId` of rejected message.	Until new order is accepted.

Error Type	Error level	Description	Reference	Report duration
‘OUTSIDE_OF_CORRIDOR’	‘CRITICAL’	Leaving the corridor defined for an edge.	`edgeId`	Until the mobile robot is no longer violating the corridor boundaries.
‘INSUFFICIENT_MEMORY’	‘URGENT’	Mobile robot does not have enough memory to process received order.	If possible, `orderId` and `orderUpdateId` of rejected message.	Until new order is accepted.
‘DUPLICATE_MAP’	‘WARNING’	Receival of a map with `mapId` and `mapVersion` already existing.	`mapId` and `mapVersion` of duplicate	Until a new map related instantAction was accepted.
‘BLOCKED_ZONE_VIOLATION’	‘CRITICAL’	Entering a ‘BLOCKED’ zone.	`zoneId`	Until the mobile robot is no longer violating the blocked zone.
‘DUPLICATE_ZONE_SET’	‘WARNING’	Receival of a zone set with `zoneSetId` already existing.	`zoneSetId` or `actionId` of `instantAction`	Reasonable amount of time for the fleet control to notice that the zone update failed.
‘RELEASE_LOST’	‘CRITICAL’	Losing the release for a ‘RELEASE’ zone.	`zoneId`	Until the mobile robot is no longer within the ‘RELEASE’ zone or is granted a the release again.
‘ZONE_ACTION_CONFLICT’	‘CRITICAL’	Conflict between zone behavior and zone actions.	`zoneId` of ‘ACTION’ zone	Until the mobile robot is no longer violating the zone behavior.
‘NODE_UNREACHABLE’	‘CRITICAL’	The mobile robot cannot reach a node in its order.	`nodeId`	Until new order is accepted.
‘LOCALIZATION_ERROR’	‘FATAL’	The mobile robot is not localized.		Until localization is regained.
‘NO_ROUTE_TO_TARGET’	‘WARNING’	Receival of an order with at least one unreachable node.	`orderId`	Until new order is accepted.

Error Type	Error level	Description	Reference	Report duration
‘OTHER_ORDER_ACTIVE’	‘WARNING’	Receival of a new order while another order is still active.	`orderId`	Until new order is accepted.
‘START_NODE_OUT_OF_RANGE’	‘WARNING’	Receival of an order with unreachable first node.	`orderId`	Until new order is accepted.
‘MOBILE_ROBOT_NOT_AVAILABLE’	‘WARNING’	Receival of an order while not in ‘AUTOMATIC’, ‘SEMIAUTOMATIC’ or ‘INTERVENED’ operating mode.	`orderId`	Until operating mode allows for new orders
‘UNKNOWN_MAP_ID’	‘WARNING’	Receival of an order containing nodes referencing an unknown `mapId`.	`orderId`	Until new order is accepted.

Table 9: Predefined error types

6.6.6 Operating Mode

For regular order execution, fleet control shall be in full control of the mobile robot. There are however situations where this is not possible, e.g., when manual interaction on the mobile robot is required. The mobile robot shall report this using the field `operatingMode`.

The following lists describe the values of the field `operatingMode`, their meaning, and implications on the interaction between mobile robot and fleet control:

Operating Mode	Description
AUTOMATIC	Fleet control is in full control of the mobile robot. Mobile robot moves and executes actions based on orders from the fleet control.
SEMIAUTOMATIC	Fleet control is in control of the mobile robot. Mobile robot moves and executes actions based on orders from the fleet control. The driving speed is controlled by the HMI.The steering is under automatic control.
INTERVENED	Fleet control is not in control of the mobile robot. The mobile robot is reporting its state correctly.HMI can be used to control the steering, velocity and handling devices of the mobile robot.Fleet control is allowed to send orders or order updates to the mobile robot to be executed after changing back into operating mode ‘AUTOMATIC’ or ‘SEMI-AUTOMATIC’. Fleet control shall not send any instant action except `cancelOrder`.The mobile robot shall not clear the order but shall remove all zone requests from the state, also if the mobile robot is already inside a ‘RELEASE’ zone. (*Remark: If necessary, the fleet control can continue to track the position of the mobile robot and decide whether clearance for other mobile robots is possible.*) The mobile robot shall not request any permissions to enter a ‘RELEASE’ zone or for replanning inside a ‘COORDINATED_REPLANNING’ zone.If entering operating mode ‘INTERVENED’ has any impact on running actions the mobile robot shall reflect this in the state message accordingly.If the mobile robot leaves this operating mode and does not directly switch into ‘AUTOMATIC’ or ‘SEMI-AUTOMATIC’ mode it shall act according to new operating mode. If the mobile robot leaves this operating mode and switches directly into ‘AUTOMATIC’ or ‘SEMI-AUTOMATIC’ mode the mobile robot shall continue executing any current order. If the mobile robot detects during operating mode ‘INTERVENED’ that a continuation of the current order is not possible the mobile robot shall switch into operating mode ‘MANUAL’ and act accordingly.
MANUAL	Fleet control is not in control of the mobile robot. Fleet control shall not send orders or actions to the mobile robot. HMI can be used to control the steering, velocity and handling devices of the mobile robot.The position of the mobile robot is sent to the fleet control.When the mobile robot enters this mode, it immediately clears any current order.If, while being in this mode, the mobile robot detects that it is being moved to a position where the current value of `lastNodeId` cannot be used as a start node of a new order, it shall set `lastNodeId` to an empty string (““).
STARTUP	Fleet control is not in control of the mobile robot. The mobile robot is starting up and not ready to receive orders. State message parameters may be incomplete or invalid until startup is finished.
SERVICE	Fleet control is not in control of the mobile robot. Fleet control shall not send orders or actions to the mobile robot. When the mobile robot enters this mode, it immediately clears any current order.The mobile robot shall set `lastNodeId` to an empty string (““).Authorized personnel can reconfigure the mobile robot.

Operating Mode	Description
TEACH_IN	Fleet control is not in control of the mobile robot. Fleet control shall not send orders or actions to the mobile robot. When the mobile robot enters this mode, it immediately clears any current order.The mobile robot shall set `lastNodeId` to an empty string (“”).The mobile robot is being taught, e.g., mapping is done by an operator.

Table 10: Operating modes of the mobile robot

Operating Mode	Fleet Control in control	Valid state message content	Clear order when entering	Set `lastNodeId` to empty	Clear zone requests when entering	Sending instant actions allowed	Sending orders allowed
AUTOMATIC	YES	YES	NO	NO	NO	YES	YES
SEMIAUTOMATIC	YES	YES	NO	NO	NO	YES	YES
INTERVENED	NO	YES	NO	NO	YES	Only `cancelOrder` allowed	YES
MANUAL	NO	YES	YES	YES, if continuation of order is not possible	YES	NO	NO
STARTUP	NO	NO	YES	YES	YES	NO	NO
SERVICE	NO	YES	YES	YES	YES	NO	NO
TEACH_IN	NO	YES	YES	YES	YES	NO	NO

Table 11: Overview of operating modes and their implications

6.6.7 Clearing the order on the mobile robot

In response to one of the following events, the mobile robot shall stop executing the current order:

- The mobile robot is changing the operating mode to ‘MANUAL’, ‘STARTUP’, ‘SERVICE’ or ‘TEACH_IN’ (see also 6.6.6 Operating Mode).
- The mobile robot receives a `cancelOrder` instant action from fleet control.
- The mobile robot receives a `startHibernation` instant action.

In these cases the mobile robot shall clear its current order which means that:

- Any scheduled actions in the `actionStates` shall be cancelled and be reported as ‘FAILED’ in `actionStates`.
- Any running action in the `actionStates` that
  - can be cancelled (cancelAllowed = true) shall be cancelled and be reported as ‘FAILED’ in `actionStates`.
  - cannot be cancelled (cancelAllowed = false) shall be reflected by reporting ‘RUNNING’ while being executed, and afterwards as the respective state (‘FINISHED’ if successful, ‘FAILED’ otherwise).
- The value of `orderId`, `orderUpdateId`, `lastNodeId` and `lastNodeSequenceId` remain unchanged.
- The arrays `nodeStates` and `edgeStates` are set to empty lists.
- Any requests shall be removed from the state.

As long as the actions of an order are not in state ‘FINISHED’ or ‘FAILED’ the mobile robot shall not report operating mode ‘MANUAL’, ‘SERVICE’ or ‘TEACH_IN’. `nodesStates` and `edgeStates` shall not be emptied before the operating mode ‘MANUAL’, ‘SERVICE’ or ‘TEACH_IN’ is reported.

An order cancellation can only be triggered by fleet control.

6.6.8 Idle state of the mobile robot

A mobile robot is idle if its `nodeStates` and `edgeStates` are empty and all actions in the `actionStates` are either ‘FINISHED’ or ‘FAILED’. A new order shall only be accepted if the mobile robot is idle. An order update can be accepted when the mobile robot is idle or during order execution. When idle, a mobile robot can execute instantActions.

6.6.9 Action states

When a mobile robot receives an `action` as part of the order (attached to a `node` or `edge` of an order), it shall report this `action` with an `actionState` in its `actionStates` array. When a mobile robot receives an `instantAction`, it shall report this `action` with an `actionState` in its `instantActionStates` array. When a mobile robot executes a `zoneAction`, it shall report this `action` with an `actionState` in its `zoneActionStates` array. Optionally, a mobile robot can report any planned `zoneAction` here.

The current stage of an action shall be reflected in the field `actionStatus` of the corresponding `actionState` (see Table 2).

actionStatus	Description
‘WAITING’	Action was received by the mobile robot but the corresponding node was not yet traversed or the corresponding edge was not yet entered.
‘INITIALIZING’	Action was triggered, preparatory measures are initiated.
‘RUNNING’	The action is running.
‘PAUSED’	The action is paused because of a pause instantAction or external trigger (pause button on the mobile robot)
‘RETRIABLE’	Actions that failed, but can be retried, specified by the retriable parameter in the action of an order. Transition from this state is triggered by a retry or skipRetry instantAction or an external trigger.
‘FINISHED’	The action is finished. A result is reported via the `actionResult`.
‘FAILED’	Action could not be finished for whatever reason.

Table 12: Feasible values for the actionStatus field

All possible action state transitions are visualized in Figure 21 and examples are given in the following matrix:

from / to →	WAITING	INITIALIZING	PAUSED	RUNNING	RETRIABLE	FAILED	FINISHED
Initial state	Queued for later execution	starts initialization immediately (e.g., instantAction)	-	starts execution immediately (e.g., instantAction)	-	instantActions failed to execute (unknown to mobile robot, invalid parameters)	action finishes immediately (e.g., setting a parameter)
WAITING	-	preparation necessary (lifting, sensor power up)	-	no preparation necessary	-	aborted via cancel, switch to manual mode	action succeeds instantly, e.g., after reaching node/edge
INITIALIZING	-	-	external trigger	initialization finished, action starting	-	initialization failed, aborted via cancel, switch to manual mode	-
PAUSED	-	external trigger	-	external trigger	-	aborted via cancelOrder, switch to manual mode	-
RUNNING	-	-	external trigger	-	action not completed successfully but is retriable	aborted via cancel, switch to manual mode, action finally failed due to not returning the desired results	action returned desired result, possible after abort via cancelOrder, if action can not be interrupted and has to finish.
RETRIABLE	-	retries action via retry, external trigger	-	retries action via retry, external trigger	-	failed via skipRetry, failed via cancelOrder, external trigger, switch to manual mode	fixed by operator via external input

Table 13: Examples for possible action state transitions

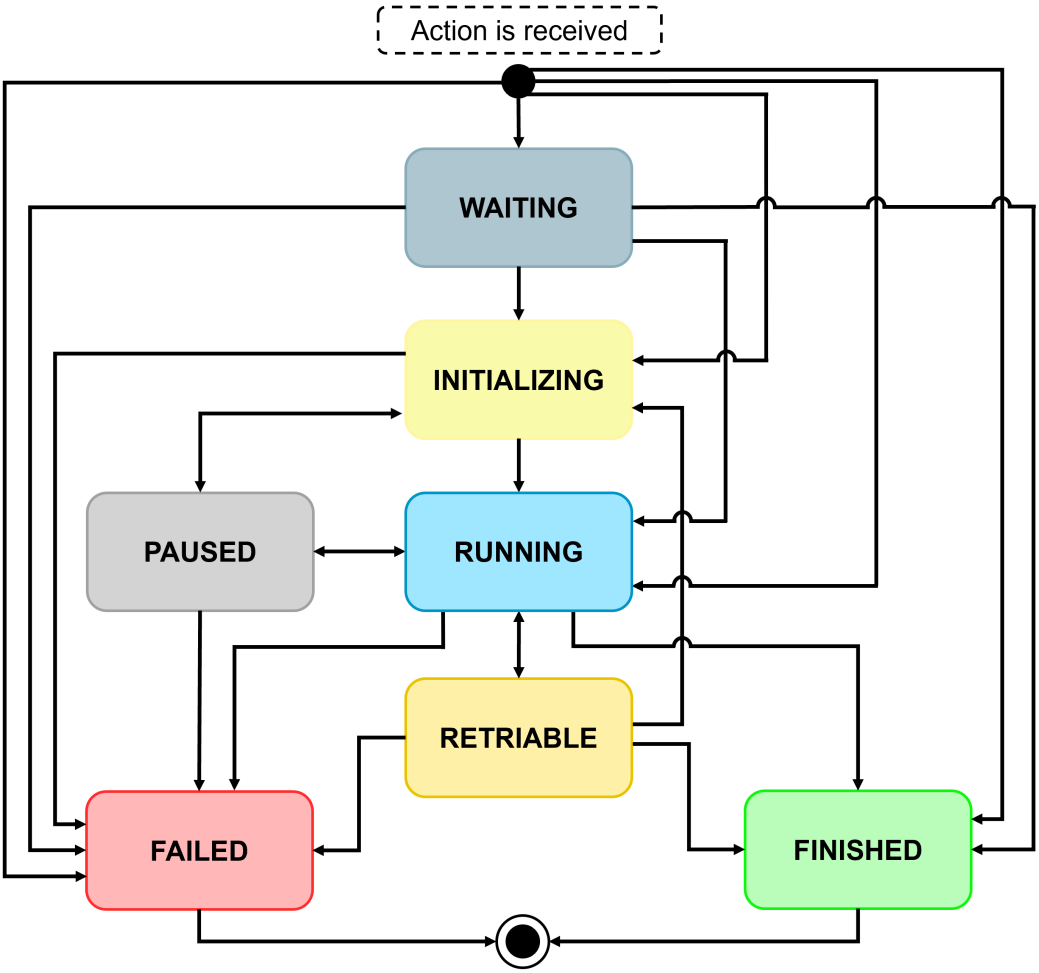


Figure 21: All possible status transitions for actionStates

6.6.9.1 Reporting of horizon actions in the mobile robot's state

The mobile robot's state shall always represent the full status of the order it currently has. Therefore, the robot shall report both the `actionsStates` of actions included in its base as well as horizon at all times. All horizon actions are reported as 'WAITING'. If the mobile robot receives an order update where part of its former horizon is removed or changed, all actions that were attached to these nodes and edges shall be removed from the `actionStates` to reflect this. `actionStates` of base actions shall never be removed in the context of an `orderUpdate` as the base cannot be modified once released.

6.6.10 Request Use of Corridors

If the corridors within a mobile robot's currently active order have the `releaseRequired` flag set to true, it shall issue a request prior to deviating from the predefined trajectory of an edge. For this purpose, the robot shall add an `edgeRequest` object to its state message. The `requestId` shall be unique across all requests (e.g., `zoneRequest`, `edgeRequest`) issued by the mobile robot.

The `requestStatus` is set to REQUESTED and the combination of `edgeId` and `sequenceId` references the edge's trajectory the robot asks to deviate from. The mobile robot has the option to request the approval for several edges simulatenously as long as they are part of its current base. The usage of each corridor shall be requested in a dedicated `edgeRequest` and each request shall be approved inidivdually by fleet control via the `response` topic (see Section 6.9 Request/response mechanism).

Fleet control shall only release the corridor for edges that are part of the base. The robot shall remain on the predefined trajectory of its current edge until a `response` is received from the fleet control. Once the robot has received the approval to start maneuvering, it sets the `requestStatus` to ‘GRANTED’ and may now use the corridor.

As long as the robot requires the corridor, it shall keep the `edgeRequest` in its state. If the mobile robot no longer requires the use of a corridor (e.g., because it might have successfully completed its avoidance procedure, no more need to avoid an obstacle, etc.), it indicates this to the fleet control by removing the corresponding `edgeRequest` object from its state. From there on, the mobile robot shall act as a line-guided mobile robot again. If it wishes to deviate from the predefined trajectory once more, it shall issue a new `edgeRequest`. If during the avoidance procedure the robot reaches the end of its current edge’s `corridor` and plans to continue to the upcoming corridor, which is not yet released, it shall stop at the border of its current `corridor`, send a dedicated edge request, and await its approval through the fleet control.

If the mobile robot’s approval expires according to the `leaseExpiry` of the response or when fleet control revokes a granted request, the mobile robot shall initiate the fallback action predefined in the `releaseLossBehavior` of the corridor of the edge. Recovery strategies for loss of release are either the mobile robot returning to the predefined trajectory of the edge along the path it took to deviate from it or stopping in its current position and awaiting manual intervention.

6.7 Visualization

For a near real-time position and planned trajectory update the mobile robot can broadcast its position, velocity and planned trajectory on the topic `visualization`.

The fields of the visualization object use the same structure as the position, velocity, planned path and intermediate path object in the state. For additional information see Implementation of the visualization message. The update rate for this topic is defined by the integrator.

6.8 Sharing of planned paths for freely navigating mobile robots

Freely navigating mobile robots shall communicate their planned trajectory to the fleet control system via the state message. For a higher frequency of sharing, the `visualization` topic can be used.

Mobile robots share their `intermediatePath`, which represents the estimated time of arrival at closer waypoints that the mobile robot is able to perceive with its sensors, and their `plannedPath`, which represents a longer path within the mobile robot’s currently active order. Both paths shall start from the mobile robot’s current position, independent of any nodes that are part of the order. The mobile robot can decide on the length of the shared paths, as it may be situation dependent. If the mobile robot is freely navigating, both `intermediatePath` and `plannedPath` shall be shared in each state. - The `plannedPath` is defined as NURBS as defined in the `trajectory` field of the `edgeState`. The `plannedPath` can contain an array of nodes, referenced by their `nodeId`, that will be traversed as part of the current path. It should be updated whenever a significant change has occurred in the mobile robot’s `plannedPath`. The `plannedPath` shall at least cover the mobile robot’s current base. - The `intermediatePath` is defined as a polyline. The polyline consists of linear line segments between waypoints. Each `waypoint` consists of its `x` and `y` position, an optional orientation of the mobile robot and the `ETA` indicating the estimated time of arrival. The `intermediatePath` shall be updated with every sent state or visualization message and always begin at the mobile robot’s current position.

The parameters `plannedPath` and `intermediatePath` shall be used only for trajectories planned by the mobile robot. The trajectory fields in the `edgeState` shall only be used to ‘acknowledge’ trajectories that have already been defined a priori within a layout or the order.

6.9 Request/response mechanism

Certain coordination tasks between mobile robots and the fleet control require explicit permission from the fleet control before the mobile robot is allowed to perform an operation. For these cases, a request/response mechanism is used. The lifecycle of a request is described in figure 22.

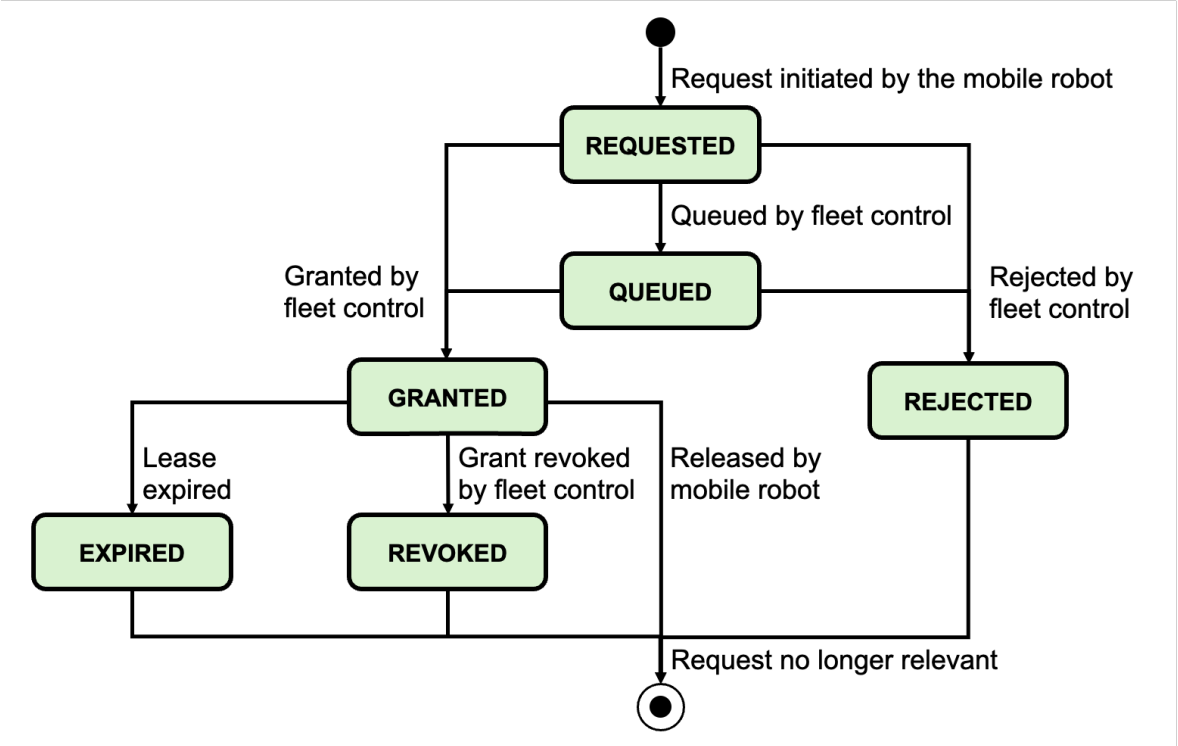


Figure 22: Request lifecycle: request states and logic of possible transitions

A request is always initiated by the mobile robot and communicated as part of the state message. The fleet control shall evaluate the request and return its decision via the `responses` topic.

Each request shall be represented on the mobile robot by a request object (e.g., zoneRequest) included in the state message. The request object shall contain at minimum:

```
- a `requestId` that is unique per mobile robot for all currently active
requests,
- a `requestType` that specifies the kind of operation the request refers
to (access, replanning, use of corridor),
- a reference to the resource the request addresses (e.g. zone, zone set,
map, edgeId, sequenceId), and
- a `requestStatus`.
```

The field `requestStatus` describes the life cycle of the request and shall support the following values:

- 'REQUESTED': Mobile robot states a request.
- 'GRANTED': The fleet control grants the request.
- 'REVOKED': Fleet control revokes previously granted request.
- 'EXPIRED': request has expired.
- ' QUEUED': Acknowledge the mobile robot's request to the fleet control, but no permission is given yet. Request was added to some sort of a queue. Fleet control receives requests from the state topic and shall answer via the `responses` topic containing a response object that includes:

- The requestId of the corresponding request,
- a decision with one of the values ‘GRANTED’, ‘QUEUED’, ‘REJECTED’, or ‘REVOKED’, and
- optionally a leaseExpiry timestamp that limits the validity of a ‘GRANTED’ decision.

If a request is answered with ‘QUEUED’, fleet control acknowledges reception of the request but does not yet grant permission. The mobile robot shall then continue to wait and shall not perform the requested operation. If a request is answered with ‘REJECTED’, the mobile robot shall not perform the requested operation and may remove the corresponding request object from its state when it is no longer needed.

If a request is answered with ‘GRANTED’, the mobile robot is allowed to perform the requested operation in accordance with the semantics of the request type. If a `leaseExpiry` is present, the permission shall only be considered valid until this time. Fleet control can extend a lease by sending an updated response with the same `requestId` and a new `leaseExpiry`.

If a request is answered with ‘REVOKED’, or if the `leaseExpiry` is reached, the mobile robot shall act according to the `releaseLossBehavior` defined for the requested resource. If the requested operation was already started, the mobile robot shall update the `requestStatus` accordingly (‘REVOKED’ or ‘EXPIRED’) and keep it in its state until the `releaseLossBehavior` is finished. If the requested operation was not started, the mobile robot shall remove the request from its state.

If no response is received within the time frame required by the application, the mobile robot shall behave as if the request had not been granted and shall not perform the operation that requires explicit permission. The handling of timeouts and retries shall be defined during integration.

A request shall be removed from the mobile robot’s state once the corresponding operation has been completed, aborted, or rejected and no further decision from fleet control is required.

6.10 Factsheet

The factsheet provides basic information about a specific mobile robot type series. This information allows comparison of different mobile robot types and can be applied for the planning, dimensioning, simulation or integration of a mobile robot system.

The values for some fields in the mobile robot factsheet can only be specified during system integration, for example the assignment of project-specific load supported by a mobile robot. The factsheet is intended as both a human-readable document and for machine processing, e.g., an import by the fleet control application, and thus is specified as a JSON document.

The fleet control can request the factsheet from the mobile robot by sending the instant action `factsheetRequest`.

All messages on this topic shall be sent with a `retained` flag.

7 Message specification

The different messages are presented in tables describing the contents of the fields of the JSON.

In addition, JSON schemas are available for validation in the public git repository (https://github.com/VDA5050/VDA5050). The JSON schemas are updated with every release of the VDA5050. If there are differences between the JSON schemas and this document, the variant in this document applies.

7.1 Symbols of the tables and meaning of formatting

The object structure tables contain the name of the identifier, its unit, its data type, and a description, if any.

Identification	Description
standard	Variable is an elementary data type
bold	Variable is a non-elementary data type (e.g., JSON object or array) and defined separately
italic	Variable is optional
italic and bold	Variable is optional and a non-elementary data type
arrayName[arrayDataType]	Variable (here arrayName) is an array of the data type included in the square brackets (here the data type is arrayDataType)

Table 14: Symbols of the tables and meaning of formatting

All keywords are case sensitive. All field names are in camelCase.

7.1.1 Optional fields

If a variable is marked as optional, it is optional for the sender as the variable might not be applicable in certain cases (e.g., when the fleet control sends an order to a mobile robot, some mobile robots plan their trajectory themselves and the field `trajectory` within the `edge` object of the order can be omitted).

If the mobile robot receives a message that contains a field which is marked as optional in this protocol, the mobile robot is expected to act accordingly and shall not ignore the field. If the mobile robot cannot process the order due to an unsupported parameter, it shall communicate this with an error of type ‘UNSUPPORTED_PARAMETER’ and error level ‘CRITICAL’ and to reject the order.

Fleet control shall only send optional fields that the mobile robot supports.

Example: Trajectories are optional. If a mobile robot cannot process trajectories, fleet control shall not send a trajectory to the mobile robot.

The mobile robot shall communicate which optional parameters it needs via a mobile robot `factsheet` message.

7.1.2 Permitted characters and field lengths

All communication is encoded in UTF-8 to enable international adaption of descriptions. The recommendation is that IDs should only use the following characters:

A-Z a-z 0-9 _ - . :

A maximum message length is not defined, but limited by the MQTT protocol specification and possibly by technical constraints defined by the factsheet.

If a mobile robot's memory is insufficient to process an incoming order, it shall reject the order and report an error of type 'INSUFFICIENT_MEMORY' with error level 'URGENT'.

The matching of maximum field lengths, string lengths or value ranges is up to the integrator.

For ease of integration, mobile robot vendors shall supply a mobile robot factsheet that is detailed in Section 7.10 Implementation of the factsheet message.

7.1.3 Notation of fields, topics and enumerations

Topics and fields in this document are highlighted in the following style: `exampleField` and `exampleTopic`. Enumerations shall be written in uppercase, using an underscore to separate words, e.g., 'EXAMPLE_ENUMERATION'. These values are enclosed in single quotation marks in the document. This includes keywords such as in the `actionStatus` field ('WAITING', 'FINISHED', etc.). An extensible enum includes but is not limited to the predefined values for the paramter.

7.1.4 JSON data types

Where possible, JSON data types shall be used. A Boolean value is thus encoded by "true" or "false", not with an enumeration ('TRUE', 'FALSE') or magic numbers. Numerical data types are specified with type and precision, e.g., float64 or uint32. Special number values from the IEEE 754 like NaN and infinity are not supported.

7.2 Protocol header

Each JSON message starts with a header. The header consists of the following individual elements. The header is not a JSON object.

Object structure	Data type	Description
headerId	uint32	Header ID of the message. The headerId is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., "2017-04-15T11:40:03.123Z").
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Serial number of the mobile robot.

7.3 Implementation of the order message

Object structure	Unit	Data type	Description
headerId		uint32	Header ID of the message. The header ID is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp		string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., "2017-04-15T11:40:03.123Z").
version		string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2)

Object structure	Unit	Data type	Description
manufacturer		string	Manufacturer of the mobile robot.
serialNumber		string	Serial number of the mobile robot.
orderId		string	Order identification.This is to be used to identify multiple order messages that belong to the same order.
orderUpdateId		uint32	Order update identification. Shall be unique per orderId and start at 0 for a new order. If an order update is rejected, this field shall be passed in the respective error.
orderDescription		string	Additional human-readable information only for visualization purposes; this may not be used for any logical processes.
nodes [node]		array	Array of node objects to be traversed for fulfilling the order.
edges [edge]		array	Array of edge objects to be traversed for fulfilling the order.

Object structure	Unit	Data type	Description
node {		JSON object	
nodeId		string	Unique identifier of the node. The same node can be referenced multiple times within one order message. `sequenceId` is used to differentiate the sequence of traversal.
sequenceId		uint32	Number to track the sequence of nodes and edges in an order and to simplify order updates. The main purpose is to distinguish between a node, which is passed more than once within one orderId. The sequenceId is shared between nodes and edges and defines the sequence of traversal.
nodeDescriptor		string	Additional information on the node
released		boolean	“true” indicates that the node is part of the base. “false” indicates that the node is part of the horizon.
nodePosition		JSON object	Node position. Optional for mobile robot types that do not require the node position (e.g., line-guided mobile robots).
actions [action] }		array	Array of actions to be executed on a node. Empty array, if no actions required.

Object structure	Unit	Data type	Description
nodePosition {		JSON object	Defines the position on a map in a global project-specific world coordinate system. Each floor has its own map. All maps shall use the same project-specific global origin.
x	m	float64	X-position on the map in reference to the global project-specific coordinate system. Precision is up to the specific implementation.
y	m	float64	Y-position on the map in reference to the global project-specific coordinate system. Precision is up to the specific implementation.
theta	rad	float64	Range: [-Pi … Pi] Absolute orientation a mobile robot shall match on a node for it to be considered traversed. If defined, the mobile robot shall match the orientation on this node. If previous edge disallows rotation, the mobile robot shall rotate on the node. If following edge has a differing orientation defined but disallows rotation, the mobile robot shall rotate on the node to the edges desired rotation before entering the edge.
allowedDeviationXY	m	JSON object	Indicates how precisely a mobile robot shall match the position of a node for it to be considered traversed (see also Section Order cancellation and Traversal of nodes).
allowedDeviationTheta	rad	float64	Range: [0.0 … Pi] If defined, indicates how precisely a mobile robot shall match the orientation of a node for it to be considered traversed. The lowest acceptable angle is *`theta`* - *`allowedDeviationTheta`* and the highest acceptable angle is *`theta`* + *`allowedDeviationTheta`*. If `theta` is not specified no requirement exists for the mobile robot orientation. If = 0.0: no deviation is allowed, which means the mobile robot shall reach the node orientation as precisely as is technically possible for the mobile robot. This applies also if `allowedDeviationTheta` is smaller than the technical tolerance of the mobile robot. If the mobile robot supports this attribute, but it is not defined for this node by fleet control the mobile robot shall assume this value as 0.0.
mapId		string	Unique identification of the map on which the position is referenced. Each map has the same project-specific global origin of coordinates. When a mobile robot uses an elevator, e.g., leading from a departure floor to a target floor, it will disappear off the map of the departure floor and spawn in the related lift node on the map of the target floor.

Object structure	Unit	Data type	Description
allowedDeviationXY {		JSON object	Indicates how precisely a mobile robot shall match the position of a node for it to be considered traversed. If a = b= 0.0: no deviation is allowed, which means the mobile robot shall reach or pass the node position with the mobile robot control point as precisely as is technically possible for the mobile robot. This applies also if `allowedDeviationXY` is smaller than what is technically viable for the mobile robot. If the mobile robot supports this attribute, but it is not defined for this node by fleet control the mobile robot shall assume the value of a and b as 0.0. The coordinates of the node defines the center of the ellipse.
a	m	float64	Length of the ellipse semi-major axis in meters.
b	m	float64	Length of the ellipse semi-minor axis in meters.
theta }	rad	float64	Rotation angle (the angle from the positive horizontal axis to the ellipse's major axis inside the project-specific coordinate system).

Object structure	Unit	Data type	Description
action {		JSON object	Describes an action that the mobile robot can perform.
actionType		string	Type of the action. For predefined actions this is defined in the first column of table 4. Identifies the function of the action.
actionId		string	Unique ID to identify the action and map them to the `actionState` in the state. Suggestion: Use UUIDs.
actionDescriptor		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
blockingType		string	Enum {'NONE', 'SINGLE', 'SOFT', 'HARD'}: 'NONE': allows driving and other actions; 'SINGLE': allows driving but no other actions; 'SOFT': allows other actions but not driving; 'HARD': is the only allowed action at that time.
actionParameters [actionParameter]		array	Array of actionParameter objects for the indicated action, e.g., "deviceId", "loadId", "external triggers". An example implementation can be found in 7.3.1 Format of action parameters.
retriable }		boolean	"true": action can enter RETRIABLE state if it fails. "false": action enters FAILED state directly after it fails. Default: "false".

Object structure	Unit	Data type	Description
edge {		JSON object	Directional connection between two nodes.
edgeId		string	Unique identifier of the edge. The same edge can be referenced multiple times within one order message. `sequenceId` is used to differentiate the sequence of traversal.
sequenceId		uint32	Number to track the sequence of nodes and edges in an order and to simplify order updates. The sequenceId is shared between nodes and edges and defines the sequence of traversal.
edgeDescriptor		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
released		boolean	“true” indicates that the edge is part of the base.“false” indicates that the edge is part of the horizon.
maximumSpeed	m/s	float64	Permitted maximum speed on the edge. Speed is defined by the fastest measurement of the mobile robot.
maximumMobileRobot Height	m	float64	Permitted maximum height of the mobile robot, including the load, on the edge.
minimumLoadHandling DeviceHeight	m	float64	Permitted minimal height of the load handling device on the edge.
orientation	rad	float64	Orientation of the mobile robot on the trajectory of the edge. The value `orientationType` defines whether it shall be interpreted relative to the global project-specific map coordinate system or tangential to the trajectory of the edge. In case of tangential to the trajectory, 0.0 denotes driving forwards and PI denotes driving backwards. Example: orientation Pi/2 rad may lead to a rotation of 90 degrees. If the mobile robot starts in a different orientation, and if `reachOrientationBeforeEntering` is set to “false”, rotate the mobile robot on the edge to the desired orientation. If `reachOrientationBeforeEntering` is “true”, rotate before entering the edge. If this is not possible, the order shall be rejected. If no trajectory is defined, apply the orientation and any rotation to the direct path between the two connecting nodes of the edge. If no orientation is defined, the mobile robot may assume any orientation on the edge.
orientationType		string	Enum {‘GLOBAL’, ‘TANGENTIAL’}: ‘GLOBAL’: relative to the global project-specific map coordinate system, only valid for omnidirectional mobile robots. ‘TANGENTIAL’: tangential to the trajectory of the edge. Example use: for an omnidirectional mobile robot, any orientation is possible, for differential drive mobile robots, only orientations 0.0 (forward) and Pi (backward) may be possible. Default: ‘TANGENTIAL’.

Object structure	Unit	Data type	Description
direction		string	Sets direction at junctions for navigation type physical line guided mobile robots, possible values shall be pre-defined (mobile robot-individual). Examples: “left”, “right”, “straight”, “580 Hz”.
reachOrientationBeforeEntering		boolean	This parameter is only valid for omni-directional mobile robots. “true”: Desired edge orientation shall be reached before entering the edge. “false”: Mobile robot can rotate into the desired orientation on the edge. Default: “false”.
maximumRotationSpeed	rad/s	float64	Maximum rotation speedOptional: No limit, if not set.
trajectory		JSON object	Trajectory JSON object for this edge as NURBS. Defines the path, on which the mobile robot should move between the start node and the end node of the edge. Optional: Can be omitted, if the mobile robot cannot process trajectories or if the mobile robot plans its own trajectory.
length	m	float64	Length of the path from the start node to the end node. Optional: This value is used by line-guided mobile robots to decrease their speed before reaching a stop position.
corridor		JSON object	Definition of boundaries in which a mobile robot can deviate from its trajectory, e.g., to avoid obstacles.
actions [action] }		array	Array of actions to be executed on the edge. Empty array, if no actions required. An action triggered by an edge will only be active for the time that the mobile robot is traversing the edge which triggered the action.

Object structure	Unit	Data type	Description
trajectory {		JSON object	
degree		uint32	Degree of the NURBS curve defining the trajectory.Range: [1 … uint32.max] Default: 1
knotVector [float64]		array	Array of knot values of the NURBS. The size of `knotVector` is exactly `degree` + 1 larger than the size of `controlPoints`. The multiplicities of the first and last knot, both, must be `degree` + 1 (clamped NURBS). The multiplicity of knots other than the first or last knot must not be greater than `degree` (continuity). Range of knots: [0.0 … 1.0] Default: Equidistant knots from 0.0 to 1.0 with a multiplicity of `degree` + 1 for the first and last knot, and multiplicity 1 for all other knots (uniform knots).
controlPoints [controlPoint] **}**		array	Array of controlPoint objects defining the control points of the NURBS, explicitly including the start and end point (clamped NURBS).The number of control points needs to be at least `degree` + 1.

Object structure	Unit	Data type	Description
controlPoint {		JSON object	
x	m	float64	X-coordinate described in the project-specific coordinate system.
y	m	float64	Y-coordinate described in the project-specific coordinate system.
weight		float64	The weight of the control point on the curve.Range:]0.0 … float64.max] Default: 1.0
}			

Object structure	Unit	Data type	Description
corridor {		JSON object	
leftWidth	m	float64	Range: [0.0 … float64.max]Defines the width of the corridor in meters to the left related to the trajectory of the mobile robot (see Figure 10).
rightWidth	m	float64	Range: [0.0 … float64.max]Defines the width of the corridor in meters to the right related to the trajectory of the mobile robot (see Figure 10).
corridorReferencePoint		string	Defines whether the boundaries are valid for the kinematic center or the contour of the mobile robot. If not specified the boundaries are valid to the mobile robot's kinematic center. Enum { 'KINEMATIC_CENTER' , 'CONTOUR' }
releaseRequired		boolean	Optional flag that indicates whether the robot shall request approval from fleet control. Default: "false".
releaseLossBehavior }		string	Enum { 'STOP' , 'RETURN' } Defines how the robot shall behave in the case of either its release of a corridor expiring or the release being revoked by the fleet control. 'STOP': Mobile robot shall stop and await manual intervention. 'RETURN': Mobile robot shall return to the predefined trajectory of the edge it deviated from. Default: 'STOP'.

7.3.1 Format of action parameters

Parameters for errors, information and actions are designed as an array of JSON objects with key-value pairs.

Field	Data type	Description
actionParameter {	JSON object	actionParameter for the indicated action, e.g., deviceId, loadId, external triggers.
key	string	The key of the parameter.
Value }	One of: array,boolean,number, integer,string,object	The value of the parameter that belongs to the key.

Examples for the `actionParameter` of an action “someAction” with key-value pairs for stationType and loadType:

“actionParameters”:[{“key”:“stationType”, “value”: “floor”}, {“key”:“weight”, “value”: 8.5}, {“key”: “loadType”, “value”: “pallet_eu”}]

The reason for using the proposed scheme of “key”: “actualKey”, “value”: “actualValue” is to keep the implementation generic. The “actualValue” can be of any possible JSON data type, such as array, boolean, integer, number, string or even an object.

7.4 Implementation of the instantAction message

Object structure	Data type	Description
headerId	uint32	Header ID of the message. The header ID is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., “2017-04-15T11:40:03.123Z”).
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Serial number of the mobile robot.
actions [action]	array	Array of actions that need to be performed immediately and are not part of the regular order.

Object `action` is defined in 7.3 Implementation of the order message.

7.5 Implementation of the response message

Object structure/ Identifier	Data type	Description
headerId	uint32	Header ID of the message. The headerId is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., “2017-04-15T11:40:03.123Z”).
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Serial number of the mobile robot.
responses[response]	array	Array of response objects.

Object structure/ Identifier	Data type	Description
response {	JSON object	Object which contains the fleet control’s answer to a specific request.
requestId	string	Unique per mobile robot identifier within all active requests.

Object structure/ Identifier	Data type	Description
grantType	enum	Enum {'GRANTED','QUEUED','REVOKED','REJECTED'} 'GRANTED': The fleet control has granted the request. 'REVOKED': The fleet control revokes previously granted request. 'REJECTED': The Fleet control rejects a request. 'QUEUED': Acknowledge the mobile robot's request to the fleet control, but no permission is given yet. Request was added to some sort of a queue.
leaseExpiry }	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g."2017-04-15T11:40:03.123Z"). A timestamp for the release to expire shall only be sent with reponses granting a request.

7.6 Implementation of the zoneSet message

Object structure	Data type	Description
headerId	uint32	Header ID of the message. The header ID is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., "2017-04-15T11:40:03.123Z").
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Serial number of the mobile robot.
zoneSet	JSON object	The zone set.
zoneSet{	JSON object	Zone set detailing a dedicated map.
mapId	string	Globally unique identifier of the map the zone set particularizes.
zoneSetId	string	Globally unique identifier of the zone set.
zoneSetDescriptor	string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
zones[zone] }	array	Array of zone objects.

A single zone object has the following structure:

Object structure	Data type	Description
zone{	JSON object	
zoneId	string	Locally (within the zone set) unique identifier.
zoneType	string	Enum {'BLOCKED', 'LINE_GUIDED', 'RELEASE', 'COORDINATED_REPLANNING', 'SPEED_LIMIT', 'ACTION', 'PRIORITY', 'PENALTY', 'DIRECTED', 'BIDIRECTED'}, Zone type according to section 6.4.1 Zone types.

Object structure	Data type	Description
zoneDescriptor	string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
vertices[vertex]	array	Array of vertices that define the geometric shape of the zone in a counterclockwise direction.
maximumSpeed	float64	Required only for SPEED_LIMIT zone as defined in chapter 6.4.1 Zone types.
entryActions[zoneAction]	array	Required only for ACTION zone as defined in chapter 6.4.1 Zone types.
duringActions[zoneAction]	array	Required only for ACTION zone as defined in chapter 6.4.1 Zone types.
exitActions[zoneAction]	array	Required only for ACTION zone as defined in chapter 6.4.1 Zone types.
releaseLossBehavior	string	Required only for RELEASE zone as defined in chapter 6.4.1 Zone types.
priorityFactor	float64	Required only for PRIORITY zone as defined in chapter 6.4.1 Zone types.
penaltyFactor	float64	Required only for PENALTY zone as defined in chapter 6.4.1 Zone types.
direction	float64	Required only for DIRECTED and BIDIRECTED zone as defined in chapter 6.4.1 Zone types.
directedLimitation	string	Required only for a DIRECTED zone as defined in chapter 6.4.1 Zone types.
bidirectedLimitation	string	Required only for a BIDIRECTED zone as defined in chapter 6.4.1 Zone types.
}		

A `zoneAction` follows the structure of an action, except the mobile robot generates the `actionId` itself.

Object structure	Unit	Data type	Description
zoneAction {		JSON object	Describes an action that the mobile robot can perform.
actionType		string	Name of action as described in the first column of “Actions and Parameters”. Identifies the function of the action.
actionDescriptor		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
blockingType		string	Enum {‘NONE’, ‘SINGLE’, ‘SOFT’, ‘HARD’} ‘NONE’: allows driving and other actions; ‘SINGLE’: allows driving but no other actions; ‘SOFT’: allows other actions but not driving; ‘HARD’: is the only allowed action at that time.
actionParameters [actionParameter]		array	Array of actionParameter objects for the indicated action, e.g., “deviceId”, “loadId”, “external triggers”. An example implementation can be found in 7.3.1 Format of action parameters.

Object structure	Unit	Data type	Description
retriable }		boolean	“true”: action can enter RETRIABLE state if it fails. “false”: action enters FAILED state directly after it fails. Default: “false”.

The shape of each zone object is defined through a polygon, which is communicated through its vertices. A zone with less than three vertices is invalid and shall be rejected. The polygon is assumed as closed. Only simple polygons (i.e. without intersections) shall be used. The array of vertices defining a zone is provided as a list of x-y tuples in the globally defined project-specific coordinate system in a counterclockwise direction:

Object structure	Data type	Description
vertex{	JSON object	
x	float64	X-coordinate described in the project-specific coordinate system
y }	float64	Y-coordinate described in the project-specific coordinate system

7.7 Implementation of the connection message

Identifier	Data type	Description
headerId	uint32	Header ID of the message. The headerId is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., “2017-04-15T11:40:03.123Z”).
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Serial number of the mobile robot.
connectionState	string	Enum {‘ONLINE’, ‘OFFLINE’, ‘HIBERNATING’, ‘CONNECTION_BROKEN’} ‘ONLINE’: connection between mobile robot and broker is active. ‘OFFLINE’: connection between mobile robot and broker has gone offline in a coordinated way. ‘HIBERNATING’: The mobile robot enters a low-power state and stops sending state messages. A connection to the MQTT broker shall remain active. This mode is intended for power saving or communication reduction. The mobile robot can later transition to ONLINE when instructed or via a configured wake-up mechanism. ‘CONNECTION_BROKEN’: the connection between mobile robot and broker has unexpectedly ended.

7.8 Implementation of the state message

Object structure	Unit	Data type	Description
headerId		uint32	Header ID of the message. The headerId is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp		string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., “2017-04-15T11:40:03.123Z”).
version		string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer		string	Manufacturer of the mobile robot.
serialNumber		string	Serial number of the mobile robot.
maps[map]		array	Array of map objects that are currently stored on the mobile robot.
zoneSets[zoneSet]		Array of zoneSet	Array of zoneSet objects that are currently stored on the mobile robot.
orderId		string	Unique order identification of the current order or the previously finished order. The orderId is kept until a new order is received. Empty string (“”), if no previous orderId is available.
orderUpdateId		uint32	Order update identification to identify, that an order update has been accepted by the mobile robot. “0” if no previous orderUpdateId is available.
lastNodeId		string	Node ID of last reached node or, if the mobile robot is currently on a node, current node (e.g., “node7”). Empty string (“”), if no `lastNodeId` is available.
lastNodeSequenceId		uint32	Sequence ID of the last reached node or, if the mobile robot is currently on a node, Sequence ID of current node. This value is only valid if `lastNodeId` is not an empty string (""). If `lastNodeId` is an empty string (""), the value of `lastNodeSequenceId` can be arbitrary and shall be ignored.
nodeStates [nodeState]		array	Array of nodeState objects that need to be traversed for fulfilling the order(empty array if idle)
edgeStates [edgeState]		array	Array of edgeState objects that need to be traversed for fulfilling the order(empty array if idle)
plannedPath		JSON object	Represents a path within the robot’s currently active order as NURBS.
intermediatePath		JSON object	Represents the estimated time of arrival at closer waypoints that the mobile robot is able to perceive with its sensors.
mobileRobotPosition		JSON object	Current position of the mobile robot on the map.Optional: Can only be omitted for mobile robots without the capability to localize themselves, e.g., line-guided mobile robots.
velocity		JSON object	The mobile robot velocity in its coordinates.

Object structure	Unit	Data type	Description
loads [load]		array	Loads, that are currently handled by the mobile robot. Optional: If the mobile robot cannot determine the load state, this field shall be omitted completely and not be reported as an empty array. If the mobile robot can determine the load state, but the array is empty, the mobile robot is considered unloaded.
driving		boolean	“true”: indicates, that the mobile robot is driving (manual or automatic). Other movements (e.g., lift movements) are not included here. “false”: indicates that the mobile robot is not driving.
paused		boolean	“true”: the mobile robot is currently in a paused state, either because of the push of a physical button on the mobile robot or because of an instantAction. The mobile robot can resume the order. “false”: the mobile robot is currently not in a paused state.
newBaseRequest		boolean	“true”: the mobile robot is almost at the end of the base and will reduce speed, if no new base is transmitted. Trigger for fleet control to send a new base. “false”: no base update required.
zoneRequests [zoneRequest]		array	Array of zoneRequest objects that are currently active on the mobile robot. Empty array if no zone requests are active.
edgeRequests [edgeRequest]		array	Array of edgeRequest objects that are currently active on the mobile robot. Empty array if no edge requests are active.
distanceSinceLastNode	m	float64	Used by line-guided mobile robots to indicate the distance it has been driving past the lastNodeId. Distance in meters.
actionStates [actionState]		array	Contains an array of all actions from the current order. The action states are kept as long as the order remains active and cleared when accepting a new order. This may include actions from previous nodes, that are still in progress.When an action is completed, an updated state message is published with `actionStatus` set to ‘FINISHED’ and if applicable with the corresponding `resultDescription`.
instantActionStates [actionState]		array	An array of all instant action states that the mobile robot received. Instant actions are kept in the state message until action clearInstantActions is executed. The robot may throw an errorType ‘INSTANT_ACTION_STATES_FULL’ with errorLevel ‘URGENT’ if the list is becoming too long to manage. It is recommended that the fleet control always clears this list as soon as it practically can.

Object structure	Unit	Data type	Description
zoneActionStates [actionState]		array	An array of all zone action states that are in an end state or are currently running; sharing upcoming actions is optional. Zone action states are kept in the state message until action clearZoneActions is executed. If action zones are supported, this field is required. The robot may throw an errorType 'ZONE_ACTION_STATES_FULL' with errorLevel 'URGENT' if the list is becoming too long to manage. It is recommended that the fleet control always clears this list as soon as it practically can.
powerSupply		JSON object	Contains all power-supply related information.
operatingMode		string	Enum {'STARTUP', 'AUTOMATIC', 'SEMIAUTOMATIC', 'INTERVENED', 'MANUAL', 'SERVICE', 'TEACH_IN'} For additional information, see Table in Section 6.6.6 Operating Mode.
errors [error]		array	Array of error objects. All active errors of the mobile robot shall be in the array. An empty array indicates that the mobile robot has no active errors.
information [info]		array	Array of info objects. An empty array indicates, that the mobile robot has no information. This should only be used for visualization or debugging – it shall not be used for logic in fleet control.
safetyState		JSON object	Contains all safety-related information.

Object structure	Unit	Data type	Description
map{		JSON object	
mapId		string	ID of the map describing a defined area of the mobile robot's workspace.
mapVersion		string	Version of the map.
mapStatus		string	Enum {'ENABLED', 'DISABLED'} 'ENABLED': Indicates this map is currently actively used on the mobile robot. At most one map with the same `mapId` can have its status set to 'ENABLED'. 'DISABLED': Indicates this map version is currently not enabled on the mobile robot and thus could be enabled or deleted by request.
mapDescriptor }		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.

Object structure	Unit	Data type	Description
zoneSet{		JSON object	
zoneSetId		string	Unique identifier of the zone set that is currently enabled for the map. This field shall be left empty only if the mobile robot has no zones defined for the corresponding map.
mapId		string	Identifier of the corresponding map.
zoneSetStatus }		string	Enum {ENABLED, DISABLED} 'ENABLED': Indicates this zone set is currently actively used on the mobile robot. At most one zone set for each map can have its status set to 'ENABLED'. 'DISABLED': Indicates this zone set is currently not enabled on the mobile robot and thus could be enabled or deleted by fleet control.

Object structure	Unit	Data type	Description
nodeState {		JSON object	
nodeId		string	Unique identifier of the node. The same node can be referenced multiple times within one state message. `sequenceId` is used to differentiate the sequence of traversal.
sequenceId		uint32	`sequenceId` of the node to discern multiple nodes with same nodeId.
nodeDescriptor		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
released		boolean	"true" indicates that the node is part of the base. "false" indicates that the node is part of the horizon.
nodePosition }		JSON object	Node position. Optional: Fleet control has this information. Can be sent additionally, e.g., for debugging purposes.

Object structure	Unit	Data type	Description
nodePosition {		JSON object	Defines the position on a map in the project-specific coordinate system. Each floor has its own map. All maps shall use the same project-specific global origin.
x	m	float64	X-position on the map in the project-specific coordinate system. Precision is up to the specific implementation.
y	m	float64	Y-position on the map in the project-specific coordinate system. Precision is up to the specific implementation.

Object structure	Unit	Data type	Description
theta	rad	float64	Range: [-Pi … Pi] Absolute orientation a mobile robot shall match on a node for it to be considered traversed. Optional: Mobile robot can plan the path by itself. If defined, the mobile robot shall assume the theta angle on this node. If previous edge disallows rotation, the mobile robot shall rotate on the node. If following edge has a differing orientation defined but disallows rotation, the mobile robot is to rotate on the node to the edges desired rotation before entering the edge.
mapId		string	Unique identification of the map on which the position is referenced. Each map has the same project-specific global origin of coordinates. When a mobile robot uses an elevator, e.g., leading from a departure floor to a target floor, it will disappear off the map of the departure floor and spawn in the related lift node on the map of the target floor.
}			

Object structure	Unit	Data type	Description
edgeState {		JSON object	
edgeId		string	Unique identifier of the edge. The same edge can be referenced multiple times within one state message. `sequenceId` is used to differentiate the sequence of traversal.
sequenceId		uint32	`sequenceId` of the edge to discern multiple edges with same edgeId.
edgeDescriptor		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
released		boolean	“true” indicates that the edge is part of the base.“false” indicates that the edge is part of the horizon.
trajectory }		JSON object	Reports the trajectory that has been defined a priori within a layout or was sent for this edge as part of the order. The trajectory is to be communicated as NURBS and is defined in Section 7.3 Implementation of the order messageTrajectory segments start from the point, where the mobile robot enters the edge, and terminate at the point, where the mobile robot reports that the end node was traversed.

Object structure	Unit	Data type	Description
plannedPath {		JSON object	
trajectory		JSON object	The trajectory is to be communicated as a NURBS and is defined in chapter 7.3 Implementation of the order message.

Object structure	Unit	Data type	Description
traversedNodes[nodeId]		array	Array of `nodeIds` as communicated in the currently executed order that are traversed within the shared planned path.
}			

Object structure	Unit	Data type	Description
trajectory {		JSON object	
degree		uint32	Degree of the NURBS curve defining the trajectory.Range: [1 … uint32.max] Default: 1
knotVector [float64]		array	Array of knot values of the NURBS. The size of `knotVector` is exactly `degree` + 1 larger than the size of `controlPoints`. The multiplicities of the first and last knot, both, must be `degree` + 1 (clamped NURBS).The multiplicity of knots other than the first or last knot must not be greater than `degree` (continuity).Range of knots: [0.0 … 1.0] Default: Equidistant knots from 0.0 to 1.0 with a multiplicity of `degree` + 1 for the first and last knot, and multiplicity 1 for all other knots (uniform knots).
controlPoints [controlPoint]		array	Array of controlPoint objects defining the control points of the NURBS, explicitly including the start and end point (clamped NURBS).The number of control points needs to be at least `degree` + 1.
}			

Object structure	Unit	Data type	Description
controlPoint {		JSON object	
x	m	float64	X-coordinate described in the project-specific coordinate system.
y	m	float64	Y-coordinate described in the project-specific coordinate system.
weight		float64	The weight of the control point on the curve. Range:]0.0 … float64.max] Default: 1.0
}			

Object structure	Unit	Data type	Description
intermediatePath {		JSON object	
polyline[waypoint]		array	Array of end points of segments of a polyline.
}			

Object structure	Unit	Data type	Description
waypoint {		JSON object	Endpoint of a segment within a defined polyline.
x	m	float64	X-coordinate described in the project-specific coordinate system.
y	m	float64	Y-coordinate described in the project-specific coordinate system.
theta	rad	float64	Absolute orientation of the mobile robot in the project-specific coordinate system. Range: [-Pi … Pi]
eta		string	Estimated time of arrival/traversal. ETA is formatted as a `timestamp` (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., “2017-04-15T11:40:03.123Z”).
}			

Object structure	Unit	Data type	Description
mobileRobotPosition {		JSON object	Defines the position on a map in project-specific coordinates. Each floor has its own map.
x	m	float64	X-position on the map in reference to the project-specific coordinate system. Precision is up to the specific implementation.
y	m	float64	Y-position on the map in reference to the project-specific coordinate system. Precision is up to the specific implementation.
theta		float64	Range: [-Pi … Pi]Orientation of the mobile robot.
mapId		string	Unique identification of the map in which the position is referenced. Each map has the same origin of coordinates. When a mobile robot uses an elevator from a departure floor to a destination floor, it leaves the map of the departure floor and spawns on the corresponding elevator node on the map of the destination floor.
localized		boolean	“true”: Mobile robot is localized. `x`, `y`, and `theta` can be trusted. “false”: Mobile robot is not localized. `x`, `y`, and `theta` cannot be trusted. Changing to the state to “false” shall only happen if the mobile robot cannot determine its position anymore. The mobile robot shall report this state via an error (`errorType` = ‘LOCALIZATION_ERROR’, `errorLevel` = ‘FATAL’). While this is set to “false”, the mobile robot shall not resume automatic driving or continue its order.
localizationScore		float64	Range: [0.0 … 1.0] Describes the quality of the localization and can therefore be used, e.g., by SLAM mobile robots to describe how accurate the current position information is.0.0: lowest possible

Object structure	Unit	Data type	Description
			confidence1.0: highest possible confidence. Only for logging and visualization purposes.
deviationRange	m	float64	Value for the deviation range of the position in meters. Only for logging and visualization purposes.
}			

Object structure	Unit	Data type	Description
velocity {		JSON object	
vx	m/s	float64	The mobile robot's velocity in its X-direction.
vy	m/s	float64	The mobile robot's velocity in its Y-direction.
omega }	rad/s	float64	The mobile robot's turning speed around its Z-axis.

Object structure	Unit	Data type	Description
load {		JSON object	
loadId		string	Unique identification of the load (e.g., barcode or RFID).Empty field, if the mobile robot can identify the load but did not identify the load yet.Optional if the mobile robot cannot identify the load.
loadType		string	Type of load.
loadPosition		string	Indicates, which load handling/carrying unit of the mobile robot is used, e.g., in case the mobile robot has multiple spots/positions to carry loads. For example: "front", "back", "positionC1", etc. Optional for mobile robots with only one loadPosition
boundingBoxReference		JSON object	Point of reference for the location of the bounding box. The point of reference is always the center of the bounding box's bottom surface (at height = 0) and is described in coordinates of the mobile robot's coordinate system.
loadDimensions		JSON object	Dimensions of the load's bounding box in meters.
weight }	kg	float64	Range: [0.0 … float64.max]Absolute weight of the load measured in kg.

Object structure	Unit	Data type	Description
boundingBoxReference {		JSON object	Point of reference for the location of the bounding box. The point of reference is always the center of the bounding box's bottom surface (at height = 0) and is described in coordinates of the mobile robot's coordinate system.
x		float64	X-coordinate of the point of reference.

Object structure	Unit	Data type	Description
y		float64	Y-coordinate of the point of reference.
z		float 64	Z-coordinate of the point of reference.
theta }		float64	Orientation of the loads bounding box. Important for tuggers, trains, etc.

Object structure	Unit	Data type	Description
loadDimensions {		JSON object	Dimensions of the load's bounding box in meters.
length	m	float64	Absolute length (along the mobile robot's coordinate system's x-axis) of the load's bounding box.
width	m	float64	Absolute width (along the mobile robot's coordinate system's y-axis) of the load's bounding box.
height }	m	float64	Absolute height of the load's bounding box. Optional: Set value only if known.

Object structure	Data type	Description
zoneRequest {	JSON object	Request information sent by the mobile robot to fleet control.
requestId	string	Unique per mobile robot identifier within all active requests.
requestType	string	Enum {'ACCESS', 'REPLANNING'} Specifying the type of zone the request relates to. Feasible values are 'ACCESS' or 'REPLANNING'.
zoneId	string	Locally (within the zone set) unique identifier referencing the zone the request is related to.
zoneSetId	string	Due to the `zoneId` only being unique to a `zoneSet`, the `zoneSetId` is part of the request.
requestStatus	string	Enum {'REQUESTED', 'GRANTED', 'REVOKED', 'EXPIRED'} When stating a request, this is set to 'REQUESTED'. After response or update from fleet control set to 'GRANTED' or 'REVOKED'. If lease time expires set to 'EXPIRED'.
trajectory }	object	Optional for 'COORDINATED_REPLANNING' requests only with the planned trajectory through the zone.

Object structure	Data type	Description
edgeRequest {	JSON object	Request information sent by the mobile robot to fleet control.
requestId	string	Unique per mobile robot identifier within all active requests.
requestType	enum	Enum {'CORRIDOR'} Enum specifying the type of request. Set to CORRIDOR if requesting to deviate from the predefined trajectory within the defined work space.
edgeId	string	Globally unique identifier referencing the edge the request is related to.

Object structure	Data type	Description
sequenceId	uint32	Tracking number for sequence of edge within order. Required to uniquely identify the referenced edge within the order.
requestStatus }	enum	Enum {‘REQUESTED’, ‘GRANTED’, ‘REVOKED’, ‘EXPIRED’} When stating a request, this is set to ‘REQUESTED’. After response or update from fleet control set to ‘GRANTED’ or ‘REVOKED’. If lease time expires set to ‘EXPIRED’.

Object structure	Unit	Data type	Description
actionState {		JSON object	
actionId		string	Unique identifier of the action.
actionType		string	Type of the action.Optional: Only for informational or visualization purposes. Fleet control is aware of action type as dispatched in the order.
actionDescriptor		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
actionStatus		string	Enum {‘WAITING’, ‘INITIALIZING’, ‘RUNNING’, ‘PAUSED’, ‘RETRIABLE’, ‘FINISHED’, ‘FAILED’} See Section 6.6.9 Action states.
actionResult }		string	Description of the result, e.g., the result of an RFID reading. Errors will be transmitted in errors.

Object structure	Unit	Data type	Description
powerSupply {		JSON object	
stateOfCharge	%	float64	Range: [0 … 100] State of charge of the mobile robot. For permanently powered mobile robots, this field shall be 100.
batteryVoltage	V	float64	Battery voltage.
batteryCurrent	A	float64	Battery current.
batteryHealth	%	int8	Range: [0 … 100] State describing the battery’s health.
charging		boolean	“true”: charging in progress. “false”: the mobile robot is currently not charging. Shall only be reported as “false” if the robot is available to take orders.
range }	m	uint32	Range: [0 … uint32.max] Estimated distance to drive with current state of charge.

Object structure	Unit	Data type	Description
error {		JSON object	
errorType		string	Error type, extensible enumeration including the following predefined values Enum {‘UNSUPPORTED_PARAMETER’,

Object structure	Unit	Data type	Description
			‘NO_ORDER_TO_CANCEL’, ‘VALIDATION_FAILURE’, ‘INVALID_ORDER’, ‘OUTDATED_ORDER_UPDATE’, ‘SAME_ORDER_UPDATE_ID’, ‘ORDER_UPDATE_FOLLOWING_CANCEL’, ‘OUTSIDE_OF_CORRIDOR’, ‘DUPLICATE_MAP’, ‘DUPLICATE_ZONE_SET’, ‘BLOCKED_ZONE_VIOLATION’, ‘RELEASE_LOST’, ‘ZONE_ACTION_CONFLICT’, ‘NODE_UNREACHABLE’, ‘LOCALIZATION_ERROR’, ‘UNKNOWN_MAP_ID’, …}.
errorReferences [errorReference]		array	Array of references (e.g., `nodeId`, `edgeId`, `orderId`, `actionId`, etc.) to provide more information related to the error.
errorDescription		string	Verbose description providing details and possible causes of the error.
errorDescriptionTranslations[translation]		array	Array of translations of the error description. If a particular language is not included in the collection, the value of the errorDescription field, if present, shall be used as the default.
errorHint		string	Hint on how to approach or solve the reported error.
errorHintTranslations [translation]		array	Array of translations of the error hint. If a particular language is not included in the collection, the value of the errorHint field, if present, shall be used as the default.
errorLevel }		string	Enum {‘WARNING’, ‘URGENT’, ‘CRITICAL’, ‘FATAL’} ‘WARNING’: No immediate attention required, mobile robot is able to continue active order, if any, and accept order updates or new orders. ‘URGENT’: Immediate attention required, mobile robot is able to continue active order, if any, and accept order updates or new orders. ‘CRITICAL’: Immediate attention required, mobile robot is unable to continue active order, but is able to accept a new order. ‘FATAL’: User intervention is required, mobile robot is unable to continue active order, and unable to accept order updates or new orders.

Object structure	Unit	Data type	Description
errorReference {		JSON object	
referenceKey		string	Specifies the type of reference used (e.g., `nodeId`, `edgeId`, `orderId`, `actionId`, etc.).
referenceValue }		string	The value that belongs to the reference key. For example, the ID of the node where the error occurred.

Object structure	Unit	Data type	Description
translation {		JSON object	
translationKey		string	Specifies the language of the translation according to ISO 639-1.
translationValue }		string	Translation in language of translation key.

Object structure	Unit	Data type	Description
info {		JSON object	
infoType		string	Type/name of information.
infoReferences [infoReference]		array	Array of references.
infoDescriptor		string	A user-defined, human-readable name or descriptor. This shall not be used for logical purposes.
infoLevel }		string	Enum {‘DEBUG’, ‘INFO’} ‘DEBUG’: used for debugging. ‘INFO’: used for visualization.

Object structure	Unit	Data type	Description
infoReference {		JSON object	
referenceKey		string	References the type of reference (e.g., headerId, orderId, actionId, etc.).
referenceValue }		string	References the value, which belongs to the reference key.

Object structure	Unit	Data type	Description
safetyState {		JSON object	
activeEmergencyStop		string	Enum {‘MANUAL’, ‘REMOTE’, ‘NONE’} Defining what type of emergency stop has been activated: ‘MANUAL’: emergency stop shall be acknowledged manually on the mobile robot. ‘REMOTE’: facility emergency stop shall be acknowledged remotely. ‘NONE’: no emergency stop activated.
fieldViolation }		boolean	Protective field violation (e.g., by laser or bumper).“true”:field is violated“false”:field is not violated.

7.9 Implementation of the visualization message

Field	Data type	Description
headerId	uint32	Header ID of the message. The headerId is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., “2017-04-15T11:40:03.123Z”).
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Serial number of the mobile robot.
referenceStateHeaderId	uint32	Header ID of the state message this visualization message refers to.

Field	Data type	Description
plannedPath	JSON object	Represents a path within the robot's currently active order as NURBS.
intermediatePath	JSON object	Represents the estimated time of arrival at closer waypoints that the mobile robot is able to perceive with its sensors.
mobileRobotPosition	JSON object	Current position of the mobile robot on the map.
velocity	JSON object	The mobile robot velocity in mobile robot coordinates.

Objects `plannedPath`, `intermediatePath`, `mobileRobotPosition` and `velocity` are defined in 7.8 Implementation of the state message.

7.10 Implementation of the factsheet message

Field	Data type	Description
headerId	uint32	Header ID of the message. The headerId is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.fffZ (e.g., "2017-04-15T11:40:03.123Z").
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the mobile robot.
serialNumber	string	Serial number of the mobile robot.
typeSpecification	JSON object	These parameters generally specify the class and the capabilities of the mobile robot.
physicalParameters	JSON object	These parameters specify the basic physical properties of the mobile robot.
protocolLimits	JSON object	Limits for length of identifiers, arrays, strings, and similar in MQTT communication.
protocolFeatures	JSON object	Supported features of VDA5050 protocol.
mobileRobotGeometry	JSON object	Detailed definition of mobile robot geometry.
loadSpecification	JSON object	Abstract specification of load capabilities.
mobileRobotConfiguration	JSON object	Summary of current software and hardware versions on the mobile robot and optional network information.

typeSpecification

This JSON object describes general properties of the mobile robot type.

Field	Data type	Description
seriesName	string	Free text generalized series name as specified by manufacturer.
seriesDescription	string	Free text human-readable description of the mobile robot type series.
mobileRobotKinematics	string	Simplified description of the mobile robot kinematics type.

Field	Data type	Description
		Extensible enum: {‘DIFFERENTIAL’, ‘OMNIDIRECTIONAL’, ‘THREE_WHEEL’,…} ‘DIFFERENTIAL’: differential drive, ‘OMNIDIRECTIONAL’: omnidirectional mobile robot, ‘THREE_WHEEL’: three-wheel-driven mobile robot or mobile robot with similar kinematics.
mobileRobotClass	string	Simplified description of the mobile robot class. Extensible enum: {FORKLIFT, CONVEYOR, TUGGER, CARRIER, …} FORKLIFT: forklift, CONVEYOR: Mobile robot with conveyors on it, TUGGER: tugger, CARRIER: load carrier with or without lifting unit.
maximumLoadMass	float64	[kg], Maximum loadable mass.
localizationTypes	array of string	Simplified description of localization type. Extensible enum: {‘NATURAL’, ‘REFLECTOR’, ‘RFID’,‘DMC’,‘SPOT’,‘GRID’,…} NATURAL: natural landmarks, REFLECTOR: laser reflectors, RFID: RFID tags, DMC: data matrix code, SPOT: magnetic spots, GRID: magnetic grid.
navigationTypes	array of string	Array of path planning types supported by the mobile robot, sorted by priority. Extensible enum: {‘PHYSICAL_LINE_GUIDED’, ‘VIRTUAL_LINE_GUIDED’, ‘FREELY_NAVIGATING’, …} ‘PHYSICAL_LINE_GUIDED’: no path planning, the mobile robot follows physical installed paths, ‘VIRTUAL_LINE_GUIDED’: the mobile robot follows fixed (virtual) paths, ‘FREELY_NAVIGATING’: the mobile robot plans its path by itself.
supportedZones	array of string	Array of zone types supported by the mobile robot. Enum {‘BLOCKED’, ‘LINE_GUIDED’, ‘RELEASE’, ‘COORDINATED_REPLANNING’, ‘SPEED_LIMIT’, ‘ACTION’, ‘PRIORITY’, ‘PENALTY’, ‘DIRECTED’, ‘BIDIRECTED’}.

physicalParameters

This JSON object describes physical properties of the mobile robot.

Field	Data type	Description
minimumSpeed	float64	[m/s] Minimal controlled continuous speed of the mobile robot.
maximumSpeed	float64	[m/s] Maximum speed of the mobile robot.
minimumAngularSpeed	float64	[rad/s] Minimal controlled continuous rotation speed of the mobile robot.
maximumAngularSpeed	float64	[rad/s] Maximum rotation speed of the mobile robot.
maximumAcceleration	float64	[m/s²] Maximum acceleration with maximum load.
maximumDeceleration	float64	[m/s²] Maximum deceleration with maximum load.
minimumHeight	float64	[m] Minimum height of the mobile robot.
maximumHeight	float64	[m] Maximum height of the mobile robot.
width	float64	[m] Width of the mobile robot.
length	float64	[m] Length of the mobile robot.

protocolLimits

This JSON object describes the protocol limitations of the mobile robot. If a parameter is not defined or set to zero then there is no explicit limit for this parameter.

Field	Data type	Description
maximumStringLengths {	JSON object	Maximum lengths of strings.
maximumMessageLength	uint32	Maximum MQTT message length.
maximumTopicSerialLength	uint32	Maximum length of serial number part in MQTT-topics. Affected parameters: order.serialNumber instantActions.serialNumber state.SerialNumber visualization.serialNumber connection.serialNumber zoneSet.serialNumber response.serialNumber
maximumTopicElementLength	uint32	Maximum length of all other parts in MQTT topics. Affected parameters: order.timestamp order.version order.manufacturer instantActions.timestamp instantActions.version instantActions.manufacturer state.timestamp state.version state.manufacturer visualization.timestamp visualization.version

Field	Data type	Description
		visualization.manufacturer connection.timestamp connection.version connection.manufacturer zoneSet.timestamp zoneSet.version zoneSet.manufacturer response.timestamp response.version response.manufacturer
maximumIdLength	uint32	Maximum length of ID strings. Affected parameters: order.orderId node.nodeId nodePosition.mapId action.actionId edge.edgeId map.mapId zoneSet.zoneSetId zone.zoneId zoneRequest.requestId edgeRequest.requestId
idNumericalOnly	boolean	If “true”, parameters containing Ids shall contain numerical values only.
maximumLoadIdLength	uint32	Maximum length of loadId strings.
}		
maximumArrayLengths {	JSON object	Maximum lengths of arrays.
order.nodes	uint32	Maximum number of nodes per order processable by the mobile robot.
order.edges	uint32	Maximum number of edges per order processable by the mobile robot.
node.actions	uint32	Maximum number of actions per node processable by the mobile robot.
edge.actions	uint32	Maximum number of actions per edge processable by the mobile robot.
actions.actionsParameters	uint32	Maximum number of parameters per action processable by the mobile robot.
instantActions	uint32	Maximum number of instant actions per message processable by the mobile robot.
trajectory.knotVector	uint32	Maximum number of knots per trajectory processable by the mobile robot.
trajectory.controlPoints	uint32	Maximum number of control points per trajectory processable by the mobile robot.
zoneSet.zones	uint32	Maximum number of zones per `zoneSet` processable by the mobile robot.
state.nodeStates	uint32	Maximum number of nodeStates sent by the mobile robot, maximum number of nodes in base of mobile robot.

Field	Data type	Description
state.edgeStates	uint32	Maximum number of edgeStates sent by the mobile robot, maximum number of edges in base of mobile robot.
state.loads	uint32	Maximum number of load objects sent by the mobile robot.
state.actionStates	uint32	Maximum number of objects in actionStates sent by the mobile robot.
state.instantActionStates	uint32	Maximum number of objects in instantActionStates sent by the mobile robot.
state.zoneActionStates	uint32	Maximum number of objects in zoneActionStates sent by the mobile robot.
state.errors	uint32	Maximum number of errors sent by the mobile robot in one state message.
state.information	uint32	Maximum number of information sent by the mobile robot in one state message.
error.errorReferences	uint32	Maximum number of error references sent by the mobile robot for each error.
information.infoReferences	uint32	Maximum number of info references sent by the mobile robot for each information.
}		
timing {	JSON object	Timing information.
minimumOrderInterval	float32	[s], Minimum interval sending order messages to the mobile robot.
minimumStateInterval	float32	[s], Minimum interval for sending state messages.
defaultStateInterval	float32	[s], Default interval for sending state messages, *if not defined, the default value from the main document is used*.
visualizationInterval	float32	[s], Default interval for sending messages on visualization topic.
}		

protocolFeatures

This JSON object defines order handling processes, actions and parameters which are supported by the mobile robot.

Field	Data type	Description
optionalParameters [**optionalParameters**]	array	Array of supported and/or required optional parameters.Optional parameters that are not listed here are assumed to be not supported by the mobile robot.
{		
parameter	string	Full name of optional parameter, e.g., “*order.nodes.nodePosition.allowedDeviationTheta*”.
support	enum	Type of support for the optional parameter, the following values are possible: ‘SUPPORTED’: optional parameter is supported like specified. ‘REQUIRED’: optional parameter is required for proper mobile robot operation.

Field	Data type	Description
description	string	Free-form text: description of optional parameter, e.g.,
}		
mobileRobotActions [**mobileRobotAction**]	array	Array of all actions with parameters supported by this mobile robot. This includes standard actions specified in VDA5050 and manufacturer-specific actions.
{		
actionType	string	Unique type of action corresponding to action.actionType.
actionDescription	string	Free-form text: description of the action.
actionScopes	array of enum	Array of allowed scopes for using this action type. 'INSTANT': usable as instantAction. 'NODE': usable on nodes. 'EDGE': usable on edges. 'ZONE': usable as zone action. For example: ['INSTANT', 'NODE']
actionParameters *[**actionParameter**]*	array	Array of parameters an action has. If not defined, the action has no parameters. The JSON object defined here is a different JSON object than the one used in Section 7.3 Implementation of the order message within nodes and edges.
{		
key	string	Key string for parameter.
valueDataType	enum	Data type of value, possible data types are: 'BOOL', 'NUMBER', 'INTEGER', 'STRING', 'OBJECT', 'ARRAY'.
description	string	Free-form text: description of the parameter.
isOptional	boolean	"true": optional parameter.
}		
actionResult	string	Free-form text: description of the result.
blockingTypes	array of enum	Array of possible blocking types for defined action. Enum {'NONE', 'SOFT', 'SINGLE', 'HARD'}
pauseAllowed	boolean	"true": action can be paused via startPause, "false": action cannot be paused.
cancelAllowed	boolean	"true": action can be cancelled via cancelOrder, "false": action cannot be cancelled.
}		

mobileRobotGeometry

This JSON object defines the geometry properties of the mobile robot, e.g., outlines and wheel positions.

Field	Data type	Description
wheelDefinitions [wheelDefinition]	array	Array of wheels, containing wheel arrangement and geometry.
{		
type	string	Wheel type Extensible enum {'DRIVE', 'CASTER', 'FIXED', 'MECANUM', …}.
isActiveDriven	boolean	"true": wheel is actively driven.
isActiveSteered	boolean	"true": wheel is actively steered.
position {	JSON object	
x	float64	[m], x-position in mobile robot coordinate system.
y	float64	[m], y-position in mobile robot coordinate system.
theta	float64	[rad], orientation of the wheel in mobile robot coordinate system. Necessary for fixed wheels.
}		
diameter	float64	[m], nominal diameter of wheel.
width	float64	[m], nominal width of wheel.
centerDisplacement	float64	[m], nominal displacement of the wheel's center to the rotation point (necessary for caster wheels). If the parameter is not defined, it is assumed to be 0.
constraints	string	Free-form text: can be used by the manufacturer to define constraints.
}		
envelopes2d [envelope2d]	array	Array of mobile robot envelope curves in 2D, e.g., the mechanical envelopes for unloaded and loaded state, the safety fields for different speed cases.
{		
envelope2dId	string	Identifier of the envelope curve set.
vertices[vertex]	array	The envelope curve in form of a polygon. It shall be assumed as closed. Only simple polygons shall be used.
{		
x	float64	[m], X-position of polygon point.
y	float64	[m], Y-position of polygon point.
}		
description	string	Free-form text: description of envelope curve set.
}		
envelopes3d [envelope3d]	array	Array of mobile robot envelope curves in 3D.
{		
envelope3dId	string	Identifier of the envelope curve set.
format	string	Format of data, e.g., DXF.
data	JSON object	3D-envelope curve data, format specified in 'format'.

Field	Data type	Description
url	string	Protocol and URL definition for downloading the 3D-envelope curve data, e.g., ftp://xxx.yyy.com/ac4dgvhoif5tghji.
description	string	Free-form text: description of envelope curve set
}		

loadSpecification

This JSON object specifies load handling and supported load types of the mobile robot.

Field	Data type	Description
loadPositions	array of string	Array of load positions / load handling devices. This array contains the valid values for the parameter “state.loads[].loadPosition” and for the action parameter “lhd” of the actions pick and drop. *If this array does not exist or is empty, the mobile robot has no load handling device.*
loadSets [loadSet]	array	Array of load sets that can be handled by the mobile robot
{		
setName	string	Unique name of the load set, e.g., DEFAULT, SET1, etc.
loadType	string	Type of load, e.g., EPAL, XLT1200, etc.
loadPositions	array of string	Array of load positions btw. load handling devices, this load set is valid for.*If this parameter does not exist or is empty, this load set is valid for all load handling devices on this mobile robot.*
boundingBoxReference	JSON object	Bounding box reference as defined in parameter loads[] in state message.
loadDimensions	JSON object	Load dimensions as defined in parameter loads[] in state message.
maximumWeight	float64	[kg], maximum weight of load type.
minimumLoadhandlingHeight	float64	[m], minimum allowed height for handling of this load type and weight references to boundingBoxReference.
maximumLoadhandlingHeight	float64	[m], maximum allowed height for handling of this load type and weight references to boundingBoxReference.
minimumLoadhandlingDepth	float64	[m], minimum allowed depth for this load type and weight references to boundingBoxReference.
maximumLoadhandlingDepth	float64	[m], maximum allowed depth for this load type and weight references to boundingBoxReference.
minimumLoadhandlingTilt	float64	[rad], minimum allowed tilt for this load type and weight.
maximumLoadhandlingTilt	float64	[rad], maximum allowed tilt for this load type and weight.

Field	Data type	Description
maximumSpeed	float64	[m/s], maximum allowed speed for this load type and weight.
maximumAcceleration	float64	[m/s²], maximum allowed acceleration for this load type and weight.
maximumDeceleration	float64	[m/s²], maximum allowed deceleration for this load type and weight.
pickTime	float64	[s], approx. time for picking up the load
dropTime	float64	[s], approx. time for dropping the load.
description	string	Free-form text: description of the load handling set.
}		

mobileRobotConfiguration

This JSON object details the software and hardware versions running on the mobile robot, as well as a brief summary of network information.

Field	Data type	Description
versions[versionInfo]	array	Array of key-value pair objects containing software and hardware information.
key	string	Key of the software/hardware version used. (e.g., softwareVersion)
value	string	The version corresponding to the key. (e.g., v1.12.4-beta)
}		
network {	JSON object	Information about the mobile robot's network connection. The listed information shall not be updated while the mobile robot is operating.
dnsServers	array of string	Array of Domain Name Servers (DNS) used by the mobile robot.
ntpServers	array of string	Array of Network Time Protocol (NTP) servers used by the mobile robot.
localIpAddress	string	A priori assigned IP address used to communicate with the MQTT broker. Note that this IP address should not be modified/changed during operations.
netmask	string	The subnet mask used in the network configuration corresponding to the local IP address.
defaultGateway	string	The default gateway used by the mobile robot, corresponding to the local IP address.
}		
batteryCharging {	JSON object	Information about battery charging parameters.
criticalLowChargingLevel	float64	Specifies the critical charging level in percent at or below which the fleet control should only send orders that command the mobile robot to a charging station.
maximumDesiredChargingLevel	float64	Specifies the maximum desired charging level in percent.

Field	Data type	Description
minimumDesiredChargingLevel	float64	Specifies the minimum desired charging level in percent.
minimumChargingTime	uint32	Specifies the desired minimum charging time in seconds.

Bibliography

Document	Version	Description
ISO 3691-4	December 2023	Industrial Trucks Safety Requirements and Verification-Part 4: Driverless trucks and their systems
ISO 9787	May 2013	Robots and robotic devices: Coordinate systems and motion nomenclatures
ISO 639	November 2023	Language code for the representation of the world’s languages and language groups
ISO 8601	February 2019	Date and time: Representations for information interchange
LIF – Layout Interchange Format	March 2024	Definition of a format of track layouts for exchange between the integrator of the driverless transport mobile robots and a (third-party) fleet control system.

The German Association of the Automotive Industry (VDA) consolidates more than 650 manufacturers and suppliers under one roof. The members develop and produce cars and trucks, software, trailers, superstructures, buses, parts and accessories as well as new mobility offers.

We represent the interests of the automotive industry and stand for modern, future-oriented multimodal mobility on the way to climate neutrality. The VDA represents the interests of its members in politics, the media, and social groups.

We work for electric mobility, climate-neutral drives, the implementation of climate targets, securing raw materials, digitization and networking as well as German engineering. We are committed to a competitive business and innovation location. Our industry ensures prosperity in Germany: More than 780,000 people are directly employed in the German automotive industry.

The VDA is the organizer of the largest international mobility platform IAA MOBILITY and of IAA TRANSPORTATION, the world's most important platform for the future of the commercial vehicle industry.

Publisher	German Association of the Automotive Industry Behrenstraße 35, 10117 Berlin www.vda.de/en
	German Bundestag Lobby Register No.: R001243 EU Transparency Register No.: 9557 4664 768-90
Copyright	German Association of the Automotive Industry
	Reprint, also in extracts, is only permitted, if the source is stated.
Version	Version 3.0.0, March 2026