VDMA
Fördertechnik und Intralogistik

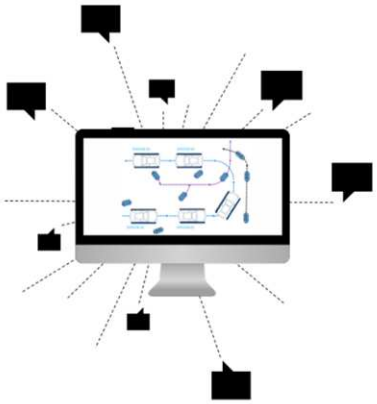
VDA German Association of the Automotive Industry

VDA Recommendation

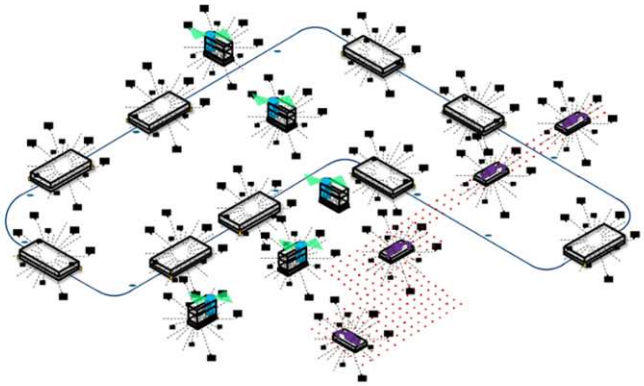
Interface for the communication between automated guided vehicles (AGV) and a master control

VDA 5050

Version 2.1.0, January 2025



control system



automated guided vehicles

Definition of a communication interface for driverless transport systems (DTS). This recommendation describes the communication interface for exchanging order and status data between a central master control and automated guided vehicles (AGVs) for intralogistics processes.

Disclaimer

The following explanations serve as an indication for the execution of an interface for communication between automated guided vehicles (AGVs) and master control and one that is freely applicable to everyone and is non-binding. Those who apply them shall ensure that they are applied properly in the specific case.

They shall take into account the state of the art prevailing at the time of each issue. By applying the proposals, no one is evasive of responsibility for their own actions. The statements do not claim to be exhaustive or to the exact interpretation of the existing legislation. They may not replace the study of relevant policies, laws and regulations. Furthermore, the special features of the respective products as well as their different possible applications shall be taken into account. Everyone acts at their own risk in this regard. Liability of the VDA and those involved in the development or application of the proposals is excluded.

If you encounter any inaccuracies in the application of the proposals or the possibility of an incorrect interpretation, please inform the VDA immediately so that any defects can be rectified.

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

Table of contents

List of Figures

List of tables

1 Foreword

The interface was established in cooperation between the Verband der Automobilindustrie e.V. (VDA) and Verband Deutscher Maschinen- und Anlagenbau e. V. (VDMA). The aim of both parties is to create a universally applicable interface. Proposals for changes to the interface shall be submitted to the VDA, are evaluated jointly with the VDMA and adopted into a new version status in the event of a positive decision. The contribution to this document via GitHub is greatly appreciated. The repository can be found at the following link: https://github.com/vda5050/vda5050.

2 Objective of the document

The objective of the recommendation is to simplify the connection of new vehicles to an existing master control system and to enable parallel operation with AGVs from different manufacturers and conventional systems (inventory systems) in the same working environment.

A uniform interface between a master control and AGVs shall be defined. This should be achieved by the following points:

- Description of a standard for communication between AGV and master control and thus a basis for the integration of transport systems into a continuous process automation using co-operating transport vehicles.
- Increase in flexibility through, among other things, increased vehicle autonomy, process modules and interface, and preferably the separation of a rigid sequence of event-controlled command chains.
- Reduction of implementation time due to high "Plug & Play" capability, as required information (e.g., order information) are provided by central services and are generally valid. Vehicles should be able to be put into operation independently of the manufacturer with the same implementation effort taking into account the requirements of occupational safety.
- Complexity reduction and increase of the "Plug & Play" capability of the systems through the use of uniform, overarching coordination with the corresponding logic for all transport vehicles, vehicle models and manufacturers.
- Increase in manufacturers' independence using common interfaces between vehicle control and coordination level.
- Integration of proprietary DTS inventory systems by implementing vertical communication between the proprietary master control and the superordinate master control (cf. Figure 1).

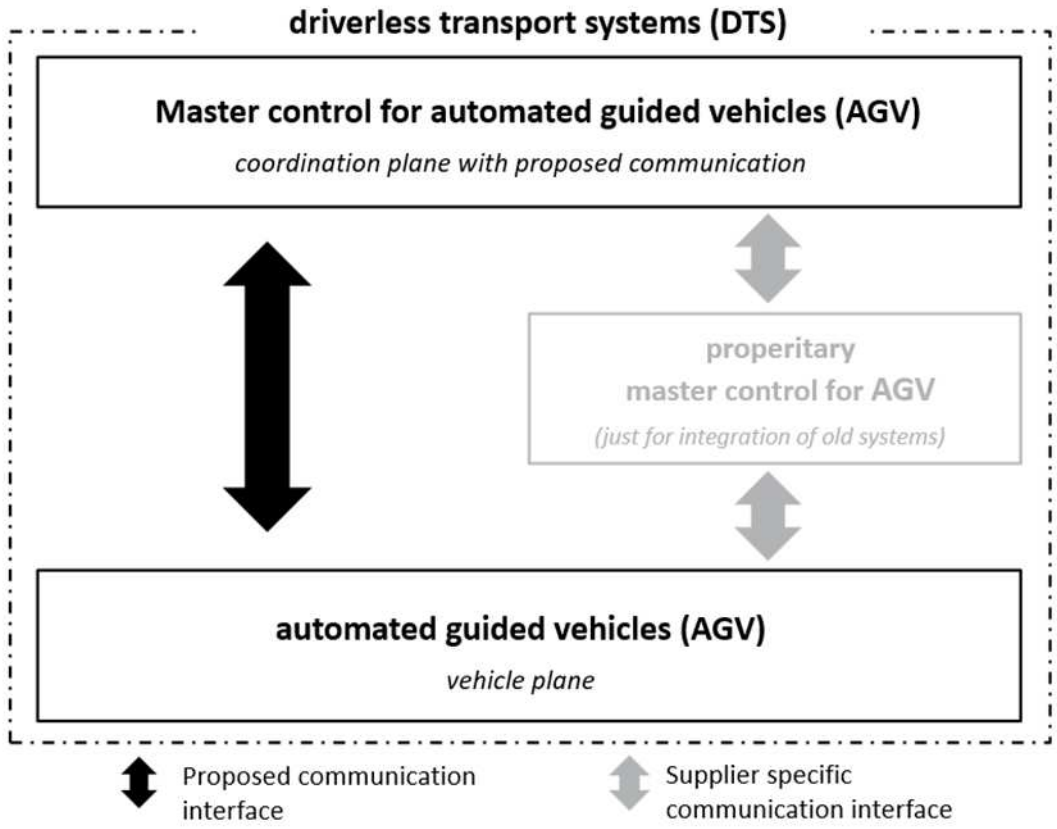


Figure 1 Integration of DTS inventory systems

In order to implement the above-mentioned objectives, this document describes an interface for the communication of order and status information between AGV and master control.

Other interfaces required for operation between AGV and master control (e.g., taking special skills freely into account with regard to path planning, etc.) or for communicating with other system components (e.g., external peripherals, fire protection gates, etc.) are not initially included in this document.

3 Scope

This recommendation contains definitions and best practice regarding communication between automated guided vehicles (AGVs) and master control. The goal is to allow AGVs with different characteristics (e.g., underrun tractor or fork lift AGV) to communicate with master control in a uniform language. This creates the basis for operating any combination of AGV in a master control. The master control provides orders and coordinates the AGVs traffic.

The interface is based on the requirements from production and plant logistics in the automotive industry. According to the formulated requirements, the requirements of intralogistics cover the requirements of the logistics department, i.e., the logistical processes from goods receiving to production supply to goods out, through control of freely navigating vehicles and guided vehicles.

In contrast to automated vehicles, autonomous vehicles solve problems that occur on the basis of the corresponding sensor system and algorithms independently and can react accordingly to changes in a dynamic environment or be adapted to them shortly afterwards. Autonomous properties such as the independent bypassing of obstacles can be fulfilled by freely navigating vehicles as well as guided vehicles. However, as soon as the path planning is carried out on the vehicle itself, this document describes freely navigating vehicles (see glossary). Autonomous systems are not completely decentralized (swarm intelligence) and have defined behavior through predefined rules.

For the purpose of a sustainable solution, an interface is described below which can be expanded in its structure. This should enable a complete coverage of the master control for vehicles that are guided. Vehicles that are freely navigating can be integrated into the structure; a detailed specification required for this is not part of this recommendation.

For the integration of proprietary stock systems, individual definitions of the interface may be required, which are not considered as part of this recommendation.

3.1 Other applicable documents

Document	Version	Description
VDI Guideline 2510	October 2005	Driverless transport systems (DTS)
VDI Guideline 4451 Sheet 7	October 2005	Compatibility of driverless transport systems (DTS) - DTS master control
DIN EN ISO 3691-4	December 2023	Industrial Trucks Safety Requirements and Verification- Part 4: Driverless trucks and their systems
LIF – Layout Interchange Format	March 2024	Definition of a format of track layouts for exchange between the integrator of the driverless transport vehicles and a (third-party) master control system.

4 Requirements and protocol definition

The communication interface is designed to support the following requirements:

- Control of min. 1000 vehicles
- Enabling the integration of vehicles with different degrees of autonomy
- Enable decision, e.g., with regard to the selection of routes or the behavior at intersections

Vehicles should transfer their status at a regular interval or when their status changes.

Communication is done over wireless networks, taking into account the effects of connection failures and loss of messages.

The message protocol is Message Queuing Telemetry Transport (MQTT), which is to be used in conjunction with a JSON structure. MQTT 3.1.1 was tested during the development of this protocol and 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 or interests them.

The JSON structure allows for a future extension of the protocol with additional parameters. The parameters are described in English to ensure that the protocol is readable, comprehensible and applicable outside the German-speaking area.

5 Process and content of communication

There are at least the following participants for the operation of AGVs:

- The operator of the AGV system provides basic information
- The master control organizes and manages the operation
- The AGV carries out the orders

Figure 2 describes the communication content during the operational phase. During implementation or modification, the AGV and master control are manually configured.

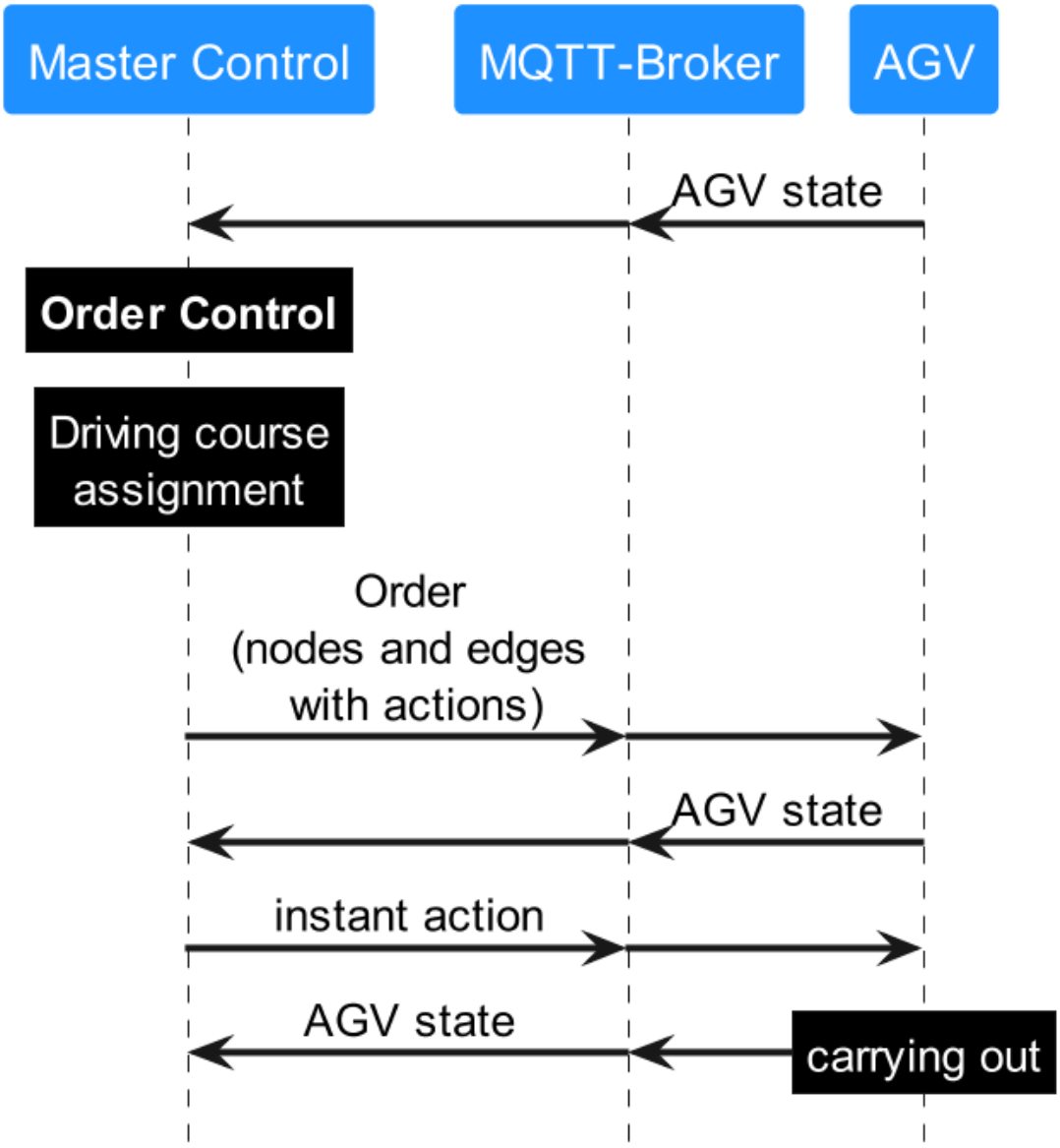


Figure 2 Structure of the Information Flow

During the implementation phase, the driverless transport systems (DTS) consisting of master control and AGVs 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 master control by importing from other systems. Essentially, this concerns the following content:

- Definition of routes: Using CAD import, routes can be imported to the master control. Alternatively, routes can also be implemented manually in the master control by the operator. Routes can be one-way streets, restricted for certain vehicle 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.
- Vehicle configuration: The physical properties of an AGV (size, available load carrier mounts, etc.) are stored by the operator. The AGV shall communicate this information via the topic `factsheet` in a specific way that is defined in Section 6.15 Topic "factsheet" 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 master control based on this information and the transport requirements to be completed. The resulting orders for an AGV are then transferred to the vehicle via an MQTT message broker. This then continuously reports its status to the master control in parallel with the execution of the job. This is also done using the MQTT message broker.

Functions of the master control are:

- Assignment of orders to the AGVs
- Route calculation and guidance of the AGVs (taking into account the limitations of the individual physical properties of each AGV, 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

Functions of the AGVs are:

- Localization
- Navigation along associated routes (guided or autonomous)
- Execution of actions
- Continuous transmission of vehicle status

In addition, the integrator shall take into account the following when configuring the overall system (incomplete list):

- Map configuration: The coordinate systems of the master control and the AGV shall be matched.
- Pivot point: The use of different points of the AGV or points of charge as a pivot point leads to different envelopes of the vehicle. The reference point may vary depending on the situation, e.g., it may be different for an AGV carrying a load and for an AGV that does not carry a load.

6 Protocol specification

The following section describes the details of the communication protocol. The protocol specifies the communication between the master control and the AGV. Communication between the AGV and peripheral equipment, e.g., between the AGV and a gate, is excluded.

The different messages are presented in tables describing the contents of the fields of the JSON that is sent as an order, state, etc.

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.

6.1 Symbols of the tables and meaning of formatting

The table contains 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)

All keywords are case sensitive. All field names are in camelCase. All enumerations are in UPPERCASE without underscores.

6.1.1 Optional fields

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

If the AGV receives a message that contains a field which is marked as optional in this protocol, the AGV is expected to act accordingly and cannot ignore the field. If the AGV cannot process the message accordingly then the expected behavior is to communicate this within an error message and to reject the order.

Master control shall only send optional information that the AGV supports.

Example: Trajectories are optional. If an AGV cannot process trajectories, master control shall not send a trajectory to the vehicle.

The AGV shall communicate which optional parameters it needs via an AGV `factsheet` message.

6.1.2 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 perhaps by technical constraints defined inside the factsheet. If an AGVs memory is insufficient to process an incoming order, it is to reject the order. The matching of maximum field lengths, string lengths or value ranges is up to the integrator. For ease of integration, AGV vendors shall supply an AGV factsheet that is detailed in 6.15.1 Factsheet JSON structure.

6.1.3 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. These values are enclosed in the document with single quotes. This includes keywords as in `actionStatus` ('WAITING', 'FINISHED', etc.).

6.1.4 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.

6.2 MQTT 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 6.14 Topic "connection".

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

Protocol security needs to be taken into account by broker configuration.

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

6.3 MQTT 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 has to be adapted individually to match the topics defined in this protocol. This means that 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:

```
uagv/v2/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 AGV.
serialNumber	string	Unique AGV serial number consisting of the following characters: A-Z a-z 0-9 _ . : -
topic	string	Topic (e.g., order or state) see Section 6.5 Topics for communication.

Note: Since the / character is used to define topic hierarchies, it shall not be used in any of the aforementioned fields. The $ character is also used in some MQTT brokers for special internal topics, so it should not be used either.

6.4 Protocol header

Each JSON message starts with a header. In the following sections, the following fields will be referenced as header for readability. The header consists of the following individual elements. The header is not a JSON object.

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.ffZ (e.g., "2017-04-15T11:40:03.12Z").
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the AGV.
serialNumber	string	Serial number of the AGV.

Protocol version

The protocol version uses semantic versioning as versioning schema.

Examples for major version changes:

- Breaking changes, e.g., new non-optional fields

Examples for minor version changes:

- New features like an additional topic for visualization

Examples for patch version:

- Higher available precision for a batteryCharge

6.5 Topics for communication

The AGV protocol uses the following topics for information exchange between master control and AGV.

Topic name	Published by	Subscribed by	Used for	Implementation	Schema
order	master control	AGV	Communication of driving orders from master control to the AGV	mandatory	order.schema
instantActions	master control	AGV	Communication of the actions that are to be executed immediately	mandatory	instantActions.schema
state	AGV	master control	Communication of the AGV state	mandatory	state.schema
visualization	AGV	visualization systems	Higher frequency of position topic for visualization purposes only	optional	visualization.schema
connection	Broker/ AGV	master control	Indicates when AGV connection is lost, not to be used by master control for checking the vehicle health, added for an MQTT protocol level check of connection	mandatory	connection.schema
factsheet	AGV	master control	Parameters or vendor-specific information to assist set-up of the AGV in master control	mandatory	factsheet.schema

6.6 Topic: "order" (from master control to AGV)

The topic "order" is the MQTT topic via which the AGV receives a JSON encapsulated order.

6.6.1 Concept and logic

The basic structure of an order is a graph of nodes and edges. The AGV is expected to traverse the nodes and edges to fulfill the order. The full graph of all connected nodes and edges is held by master control.

The graph representation in the master control contains restrictions, e.g., which AGV is allowed to traverse which edge. These restrictions will not be communicated to the AGV. The master control only includes edges in an AGV order which the concerning AGV is allowed to traverse.

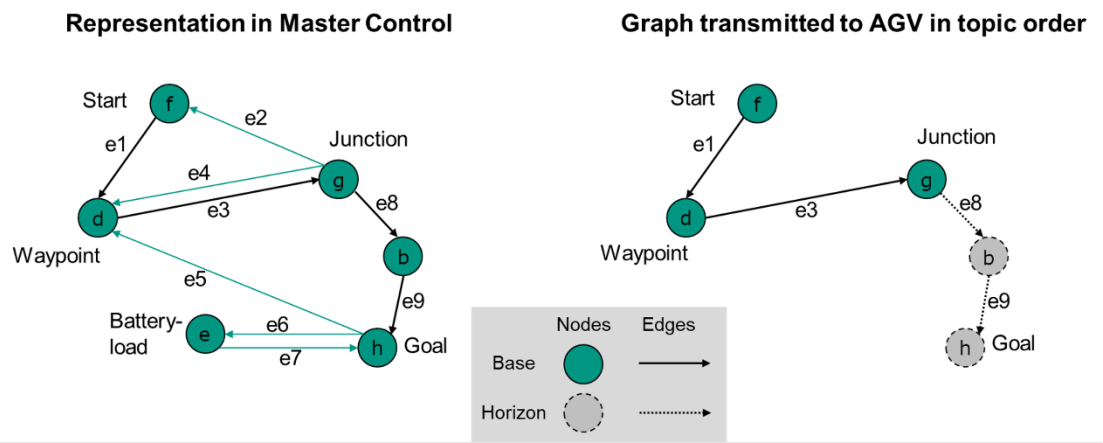


Figure 3 Graph representation in master control and graph transmitted in orders

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 in which sequence the nodes and edges shall be traversed.

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 shall be trivially reachable for the AGV. This means either that the AGV is already standing on the node, or that the AGV is in the node's deviation range.

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

6.6.2 Orders and order update

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

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

The AGV 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 master control should extend the base before the AGV 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 master control shall therefore assume that the base has already been executed by the AGV. A later section describes a procedure to cancel an order, but this is also considered unreliable due to the communication limitations mentioned above.

The master control has the possibility to change the horizon by sending an updated route to the AGV which includes the changed list of nodes and edges. The procedure for changing the horizon route is shown in Figure 4.

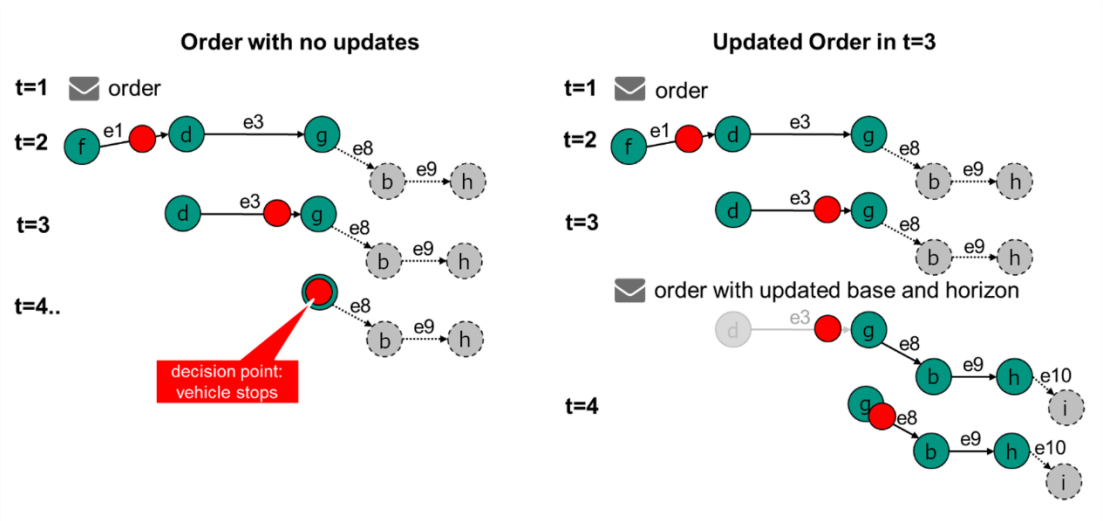


Figure 4 Procedure for changing the driving route "Horizon"

In Figure 4, an initial job is first sent by the control panel at time t = 1. Figure 5 shows the pseudocode of a possible job. 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 5 Pseudocode of an order.

At time t = 3, the order is updated by sending an extension of the order (see example in Figure 6). Note that the `orderUpdateId` is incremented and that the first node of the order update corresponds to the last shared base node of the previous order message.

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

```
{
      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 6 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 AGV can always check that the last known base node has the same `nodeId` (and `nodeSequenceId`, more on that later) as the first new base node.

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.

It is important, that the contents of the stitching node (node g in the example case) are not changed. For actions, deviation range, etc., the AGV shall use the instructions provided in the first order (Figure 5, orderUpdateId 0).

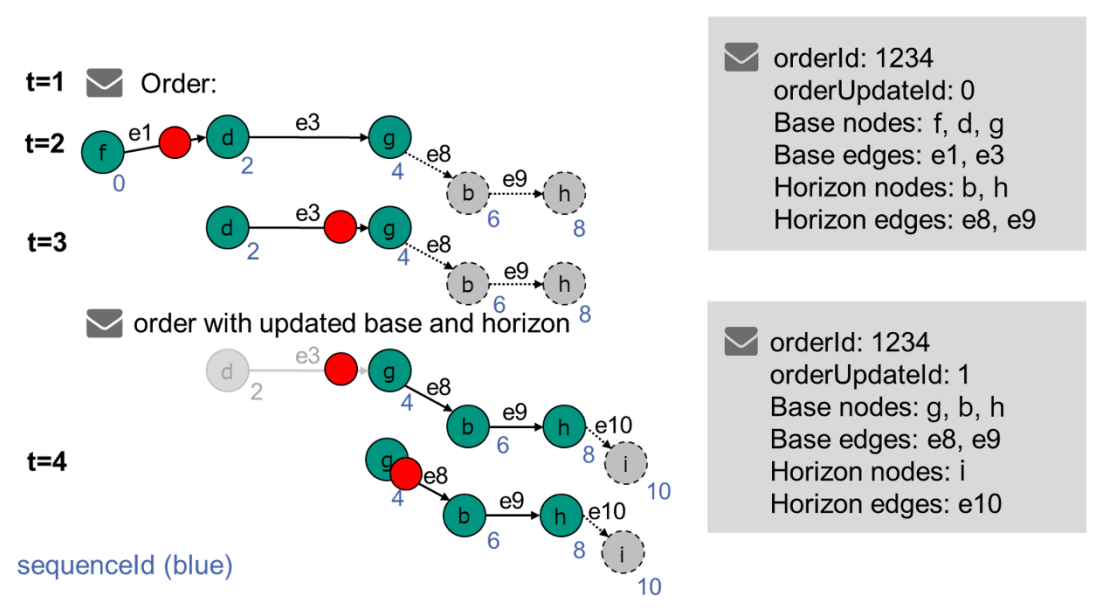


Figure 7 Regular update process - order extension.

Figure 7 describes how an order should be extended. It shows the information that is currently available on the AGV. The `orderId` stays the same and the `orderUpdateId` is incremented.

The last node of the previous base is the first base node in the updated order. With this node the AGV can add the updated order onto the current order (stitching). The other nodes and edges from the previous base are not resent.

Master control has the option to make changes to the horizon by sending entirely different nodes as the new base. The horizon can also be deleted.

To allow loops in orders (like going from node a to b and then back to a) a `sequenceId` is assigned to the node and edge objects. This `sequenceId` runs over the nodes and edges (first node of an order receives a 0, the first edge then gets the 1, the second node then gets the 2, and so on). This allows for easier tracking of the order progress.

Once a `sequenceId` is assigned, it does not change with order updates (see Figure 7). This is necessary to determine on AGV side to which node the master control refers to.

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

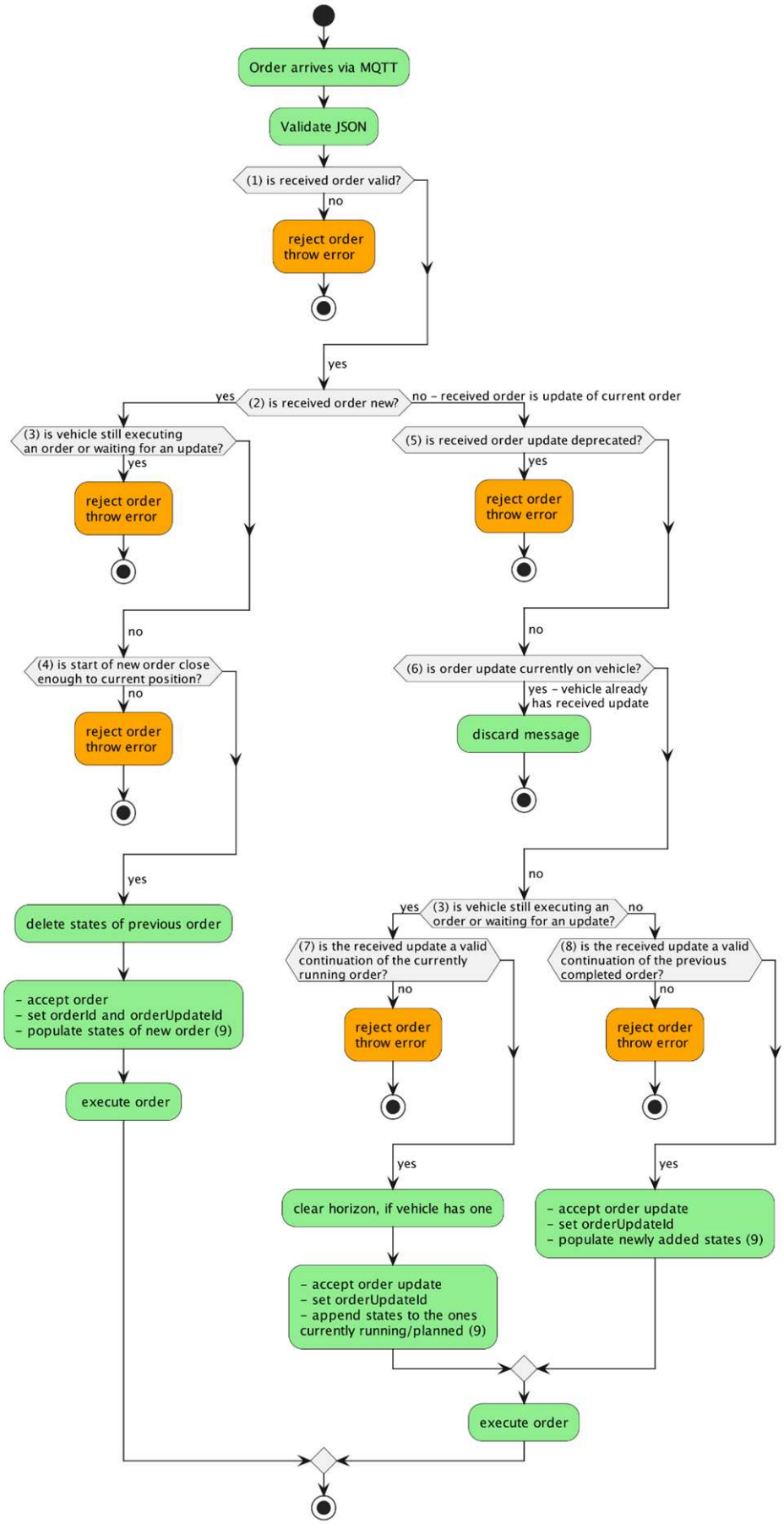


Figure 8 The process of accepting an order or order update.

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 vehicle currently holds?
3. **is vehicle still executing an order or waiting for an update?**: Are `nodeStates` not empty or are `actionStates` containing states which are neither 'FAILED' nor 'FINISHED'? Nodes and edges and the corresponding action states of the order horizon are also included inside the state. Vehicle might still have a horizon and therefore waiting for an update and executing an order.
4. **is start of new order close enough to current position?**: Is the vehicle already standing on the node, or is it in the node's deviation range (see Section 6.6.1 Concept and logic)?
5. **is received order update deprecated?**: Is `orderUpdateId` smaller than the one currently on the vehicle?
6. **is received order update currently on vehicle?**: Is `orderUpdateId` equal to the one currently on the vehicle?
7. **is the received update a valid continuation of the currently still running order?**: Is the first node of the received order equal to the current decision point (last node of the current base)? The vehicle 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.
8. **is the received update a valid continuation of the previously completed order?**: Are `nodeId` and `sequenceId` of the first node of the received order update equal to `lastNodeId` and `lastNodeSequenceId`? The vehicle 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). The order update is now accepted if it continues from the last traversed node, therefore the first node of the new base needs to match the vehicle's `lastNodeId` as well as `lastNodeSequenceId`.
9. populate/append states refers to the `actionStates` / `nodeStates` / `edgeStates`.

6.6.3 Order Cancellation (by Master Control)

In the event of an unplanned change in the base nodes, the order shall be canceled by using the instantAction `cancelOrder`.

After receiving the instantAction `cancelOrder`, the vehicle stops (based on its capabilities, e.g., right where it is or on the next node).

If there are actions scheduled, these actions shall be cancelled and report 'FAILED' in their `actionState`. If there are running actions, those actions should be cancelled and also be reported as 'FAILED'. If the action cannot be interrupted, 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 actions are running, the `cancelOrder` action shall report 'RUNNING', until all actions are cancelled/finished. After all movement of the vehicle and all of its actions are stopped, the `cancelOrder` action status shall report 'FINISHED'.

The `orderId` and `orderUpdateId` are kept.

Figure 9 shows the expected behavior for different AGV capabilities.

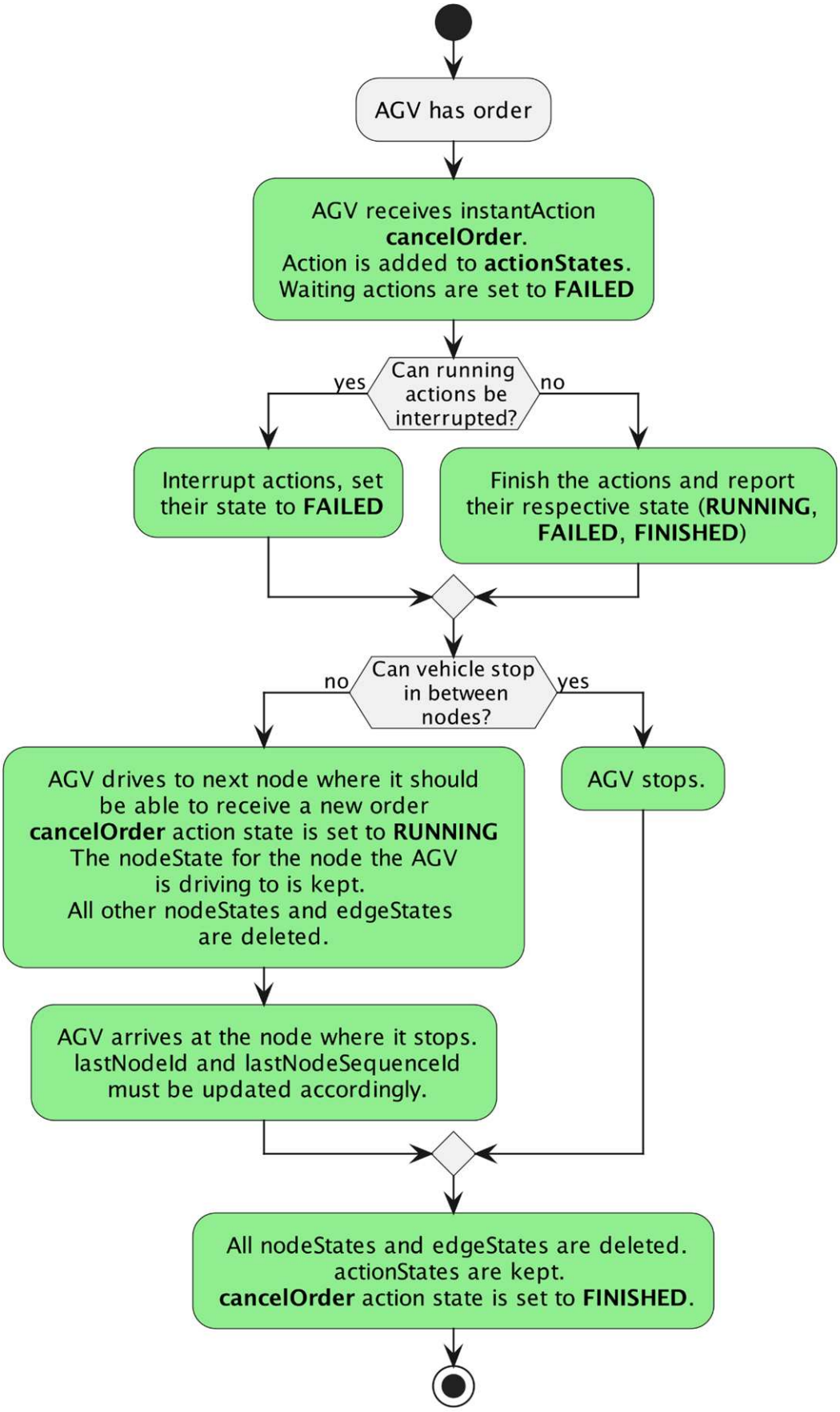


Figure 9 Expected behavior after a `cancelOrder`.

6.6.3.1 Receiving a new order after cancellation

After the cancellation of an order, the vehicle shall be in a state to receive a new order.

In the case of an AGV that localizes itself on nodes via a tag, the new order has to begin on the node the AGV is now standing on (see also Figure 5).

In case of an AGV that can stop in between nodes, the choice is up to master control how the next order should be started. The AGV shall accept both methods.

There are two options:

- Send an order, where the first node is a temporary node that is positioned where the AGV currently stands. The AGV shall then realize that this node is trivially reachable and accept the order.
- Send an order, where the first node is the last traversed node of the previous order but set the deviation range so large that the AGV is within this range. Thus, the AGV shall realize that this node shall be counted as traversed and accept the order.

6.6.3.2 Receiving a cancelOrder action when AGV has no order

If the AGV receives a `cancelOrder` action but, the AGV currently has no order, or the previous order was canceled, the `cancelOrder` action shall be reported as 'FAILED'.

The AGV shall report a “noOrderToCancel” error with the `errorLevel` set to 'WARNING'. The `actionId` of the `instantAction` shall be passed as an `errorReference`.

6.6.4 Order rejection

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

6.6.4.1 Vehicle gets a malformed new order

Resolution:

1. Vehicle does NOT take over the new order in its internal buffer.
2. The vehicle reports the warning "validationError"
3. The warning shall be reported until the vehicle has accepted a new order.

6.6.4.2 Vehicle receives an order with actions it cannot perform or with fields that it cannot use

Examples:

- Non-executable actions: lifting height higher than maximum lifting height, lifting actions although no stroke is installed, etc.
- Non-useable fields: trajectory, etc.

Resolution:

1. Vehicle does NOT take over the new order in its internal buffer
2. Vehicle reports the warning "orderError" with the wrong fields as error references
3. The warning shall be reported until the vehicle has accepted a new order.

6.6.4.3 Vehicle gets a new order with the same orderId, but a lower orderUpdateId than the current orderUpdateId

Resolution:

1. Vehicle does NOT take over the new order in its internal buffer.
2. Vehicle keeps the previous order in its buffer.
3. The vehicle reports the warning "orderUpdateError"
4. The vehicle continues with executing the previous order.

If the AGV receives an order with the same `orderId` and `orderUpdateId` twice, the second order will be ignored. This might happen, if the master control resends the order because the state message was received too late by master control and it could therefore not verify that the first order had been received.

6.6.5 Corridors

The optional `corridor` edge attribute allows the vehicle to deviate from the edge trajectory for obstacle avoidance and defines the boundaries within which the vehicle is allowed to operate.

To use the `corridor` attribute, a predefined trajectory is required that the vehicle would follow if no `corridor` attribute was defined. This can be either the trajectory defined on the vehicle known to the master control or the trajectory sent in an order. The behavior of a vehicle using the `corridor` attribute is still the behavior of a line-guided vehicle, except that it's allowed to temporarily deviate from a trajectory to avoid obstacles.

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

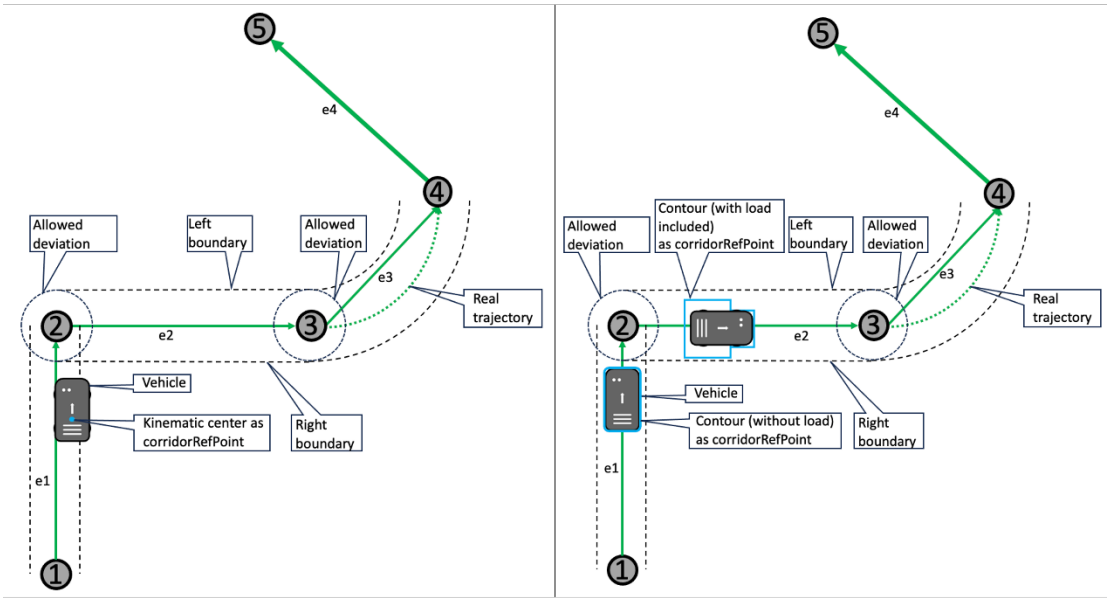


Figure 10 Edges with a `corridor` attribute that defines the left and right boundaries within which a vehicle is allowed to deviate from its predefined trajectory to avoid obstacles. On the left, the kinematic center defines the allowed deviation, while on the right, the contour of the vehicle, possibly extended by the load, defines the allowed deviation. This is defined by the `corridorRefPoint` parameter.

The area in which the vehicle is allowed to navigate independently (and deviate from the original edge trajectory) is defined by a left and a right boundary. The optional `corridorRefPoint` field specifies whether the vehicle control point or the vehicle contour should be inside the defined boundary. The boundaries of the edges shall be defined in such a way that the vehicle 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, master control shall not use the `corridor` attribute if the vehicle shall not deviate from the trajectory.

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

Remark: Allowing the vehicle to deviate from the trajectory increases the possible footprint of the vehicle during driving. This circumstance shall be considered during initial operation, and if master control makes a traffic control decision based on the vehicle's footprint.

See also Section 6.10.2 Traversal of nodes and entering/leaving edges, triggering of actions for further information.

6.6.6 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.ffZ (e.g., "2017-04-15T11:40:03.12Z")
version		string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2)
manufacturer		string	Manufacturer of the AGV
serialNumber		string	Serial number of the AGV
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. Is unique per orderId. If an order update is rejected, this field is to be passed in the rejection message.
zoneSetId		string	Unique identifier of the zone set, that the AGV has to use for navigation or that was used by master control for planning. Optional: Some master control systems do not use zones. Some AGVs do not understand zones. Do not add to message, if no zones are used.
nodes [node]		array	Array of node objects to be traversed for fulfilling the order. One node is enough for a valid order. Leave edge array empty in that case.

Object structure	Unit	Data type	Description
edges [edge]		array	Array of edge objects to be traversed for fulfilling the order. One node is enough for a valid order. Leave edge array empty in that case.

Object structure	Unit	Data type	Description
node {		JSON object	
nodeId		string	Unique node identification
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 variable sequenceId runs across all nodes and edges of the same order and is reset when a new orderId is issued.
nodeDescription		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 vehicle types that do not require the node position (e.g., line-guided vehicles).
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 map coordinate system. Precision is up to the specific implementation.
y	m	float64	Y-position on the map in reference to the map coordinate system. Precision is up to the specific implementation.
theta	rad	float64	Range: [-Pi ... Pi] Absolute orientation of the AGV on the node. Optional: vehicle can plan the path by itself. If defined, the AGV has to assume the theta angle on this node.

Object structure	Unit	Data type	Description
			If previous edge disallows rotation, the AGV shall rotate on the node. If following edge has a differing orientation defined but disallows rotation, the AGV is to rotate on the node to the edges desired rotation before entering the edge.
allowedDeviationXY	m	float64	Indicates how precisely an AGV shall match the position of a node for it to be considered traversed. If = 0.0: no deviation is allowed (no deviation means within the normal tolerance of the AGV manufacturer). If > 0.0: allowed deviation radius in meters. If the AGV passes a node within the deviation radius, the node can be considered traversed.
allowedDeviationTheta	rad	float64	Range: [0.0 … Pi] Indicates how precise the orientation defined in theta has to be met on the node by the AGV. The lowest acceptable angle is theta - allowedDeviationTheta and the highest acceptable angle is theta + allowedDeviationTheta.
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 an AGV 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.
mapDescription }		string	Additional information on the map.

Object structure	Unit	Data type	Description
action {		JSON object	Describes an action that the AGV can perform.
actionType		string	Name of action as described in the first column of "Actions and Parameters". 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.
actionDescription		string	Additional information on the action
blockingType		string	Enum {'NONE', 'SOFT', 'HARD'}: ‘NONE’: allows driving and other actions; 'SOFT': allows other actions but not driving;

Object structure	Unit	Data type	Description
			'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 7.2 Format of parameters.

Object structure	Unit	Data type	Description
edge {		JSON object	Directional connection between two nodes.
edgeId		string	Unique edge identification.
sequenceId		uint32	Number to track the sequence of nodes and edges in an order and to simplify order updates. The variable sequenceId runs across all nodes and edges of the same order and is reset when a new orderId is issued.
edgeDescription		string	Additional information on the edge.
released		boolean	"true" indicates that the edge is part of the base. "false" indicates that the edge is part of the horizon.
startNodeId		string	nodeId of first node within the order.
endNodeId		string	nodeId of the last node within the order.
maxSpeed	m/s	float64	Permitted maximum speed on the edge. Speed is defined by the fastest measurement of the vehicle.
maxHeight	m	float64	Permitted maximum height of the vehicle, including the load, on the edge.
minHeight	m	float64	Permitted minimal height of the load handling device on the edge.
orientation	rad	float64	Orientation of the AGV on the edge. The value `orientationType` defines if it has to be interpreted relative to the global project-specific map coordinate system or tangential to the edge. In case of interpreted tangential to the edge, 0.0 denotes driving forwards and PI denotes driving backwards. Example: orientation Pi/2 rad will lead to a rotation of 90 degrees. If the AGV starts in a different orientation, rotate the vehicle on the edge to the desired orientation, if `rotationAllowed` is set to "true". If `rotationAllowed` is "false", rotate before entering the edge. If that is not possible, reject the order.

Object structure	Unit	Data type	Description
			If no trajectory is defined, apply the rotation to the direct path between the two connecting nodes of the edge. If a trajectory is defined for the edge, apply the orientation to the trajectory.
orientationType		string	Enum {'GLOBAL', 'TANGENTIAL'}: 'GLOBAL': relative to the global project-specific map coordinate system; 'TANGENTIAL': tangential to the edge. If not defined, the default value is 'TANGENTIAL'.
direction		string	Sets direction at junctions for line-guided or wire-guided vehicles, to be defined initially (vehicle-individual). Examples: "left", "right", "straight".
rotationAllowed		boolean	"true": rotation is allowed on the edge. "false": rotation is not allowed on the edge. Optional: No limit, if not set.
maxRotationSpeed	rad/s	float64	Maximum rotation speed Optional: No limit, if not set.
trajectory		JSON object	Trajectory JSON object for this edge as NURBS. Defines the path, on which the AGV should move between the start node and the end node of the edge. Optional: Can be omitted, if the AGV cannot process trajectories or if the AGV 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 AGVs to decrease their speed before reaching a stop position.
corridor		JSON object	Definition of boundaries in which a vehicle can deviate from its trajectory, e.g., to avoid obstacles.
action [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 AGV is traversing the edge which triggered the action. When the AGV leaves the edge, the action

Object structure	Unit	Data type	Description
			will stop and the state before entering the edge will be restored.
trajectory {		JSON object	
degree		float64	Range: [1.0 ... float64.max] Degree of the NURBS curve defining the trajectory. If not defined, the default value is 1.
knotVector [float64]		array	Range: [0.0 … 1.0] Array of knot values of the NURBS. knotVector has size of number of control points + degree + 1.
controlPoints [controlPoint] **}**		array	Array of controlPoint objects defining the control points of the NURBS, explicitly including the start and end point.

Object structure	Unit	Data type	Description
controlPoint {		JSON object	
x		float64	X-coordinate described in the world coordinate system.
y		float64	Y-coordinate described in the world coordinate system.
weight		float64	Range: [0.0 ... float64.max] The weight of the control point on the curve. When not defined, the default will be 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 vehicle (see Figure 13).
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 vehicle (see Figure 13).
corridorRefPoint }		string	Defines whether the boundaries are valid for the kinematic center or the contour of the vehicle. If not specified the boundaries are valid to the vehicles kinematic center. Enum {'KINEMATICCENTER', 'CONTOUR'}

6.7 Maps

To ensure consistent navigation among different types of AGVs, the position is always specified in reference to the project-specific coordinate system (see Figure 11). 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 vehicle coordinate system is also specified as a right-handed coordinate system with the x-axis pointing in the forward direction of the vehicle and the z-axis pointing upward. The vehicle reference point is defined as (0,0,0) in the vehicle reference frame, unless specified otherwise. This is in accordance with Section 2.11 in DIN ISO 8855.

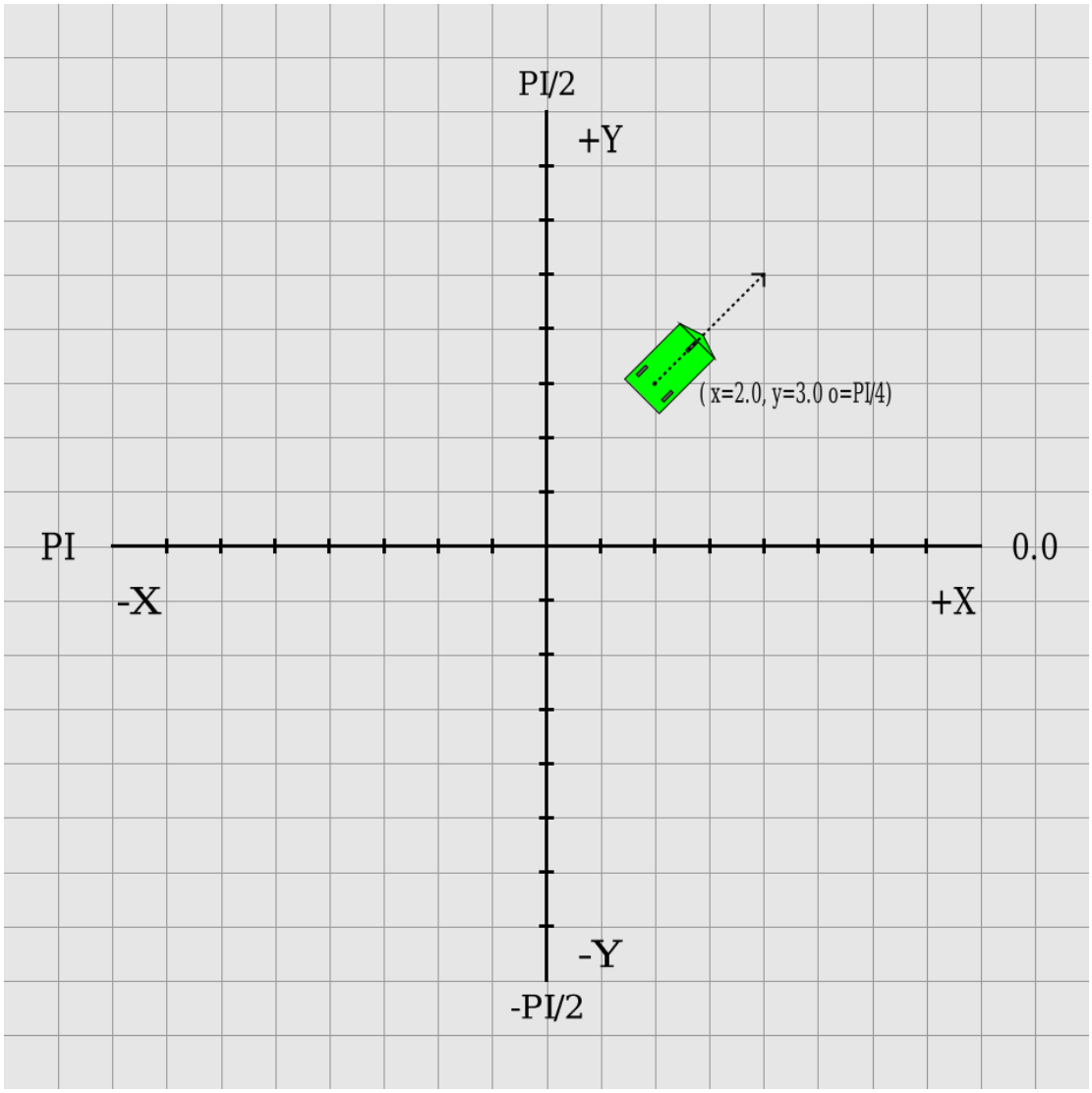


Figure 11 Coordinate system with sample AGV and orientation

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

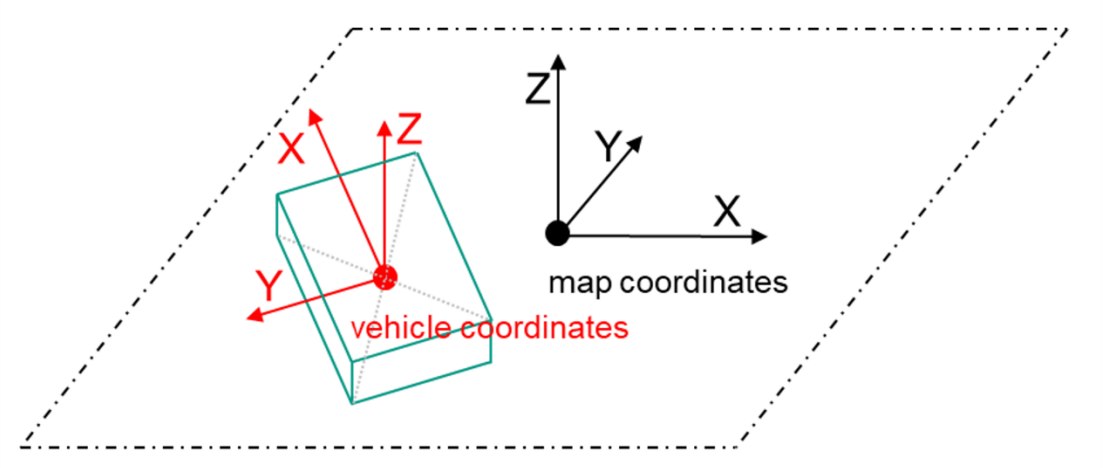


Figure 12 Coordinate systems for map and vehicle

6.7.1 Map distribution

To enable an automatic map distribution and intelligent management of restarting the vehicles if necessary, a standardized way to distribute maps is introduced.

The map files to be distributed are stored on a dedicated map server that is accessible by the vehicles. 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 vehicle is a pull operation, initiated by the master control issuing 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 vehicle's physical workspace, and the map version indicates updates to previous versions. Before accepting a new order, the vehicle shall check that there is a map on the vehicle for each map identifier in the requested order. It is the responsibility of the master control to ensure that the correct maps are activated to operate the vehicle.

In order to minimize downtime and make it easier for the master control to synchronize the activation of new maps, it is essential that maps are pre-loaded or buffered on the vehicles. The status of the maps on the vehicle can be accessed via the vehicle state channel. It's important to note that transferring a map to an AGV and then activating the map are different processes. To activate a pre-loaded map on a vehicle, the master control sends an instant action. In this case, any other map with the same map identifier but a different map version is automatically disabled. Maps can be deleted by the master control with another instant action. The result of this process is shown in the vehicle state.

The map distribution process is shown in Figure 13.

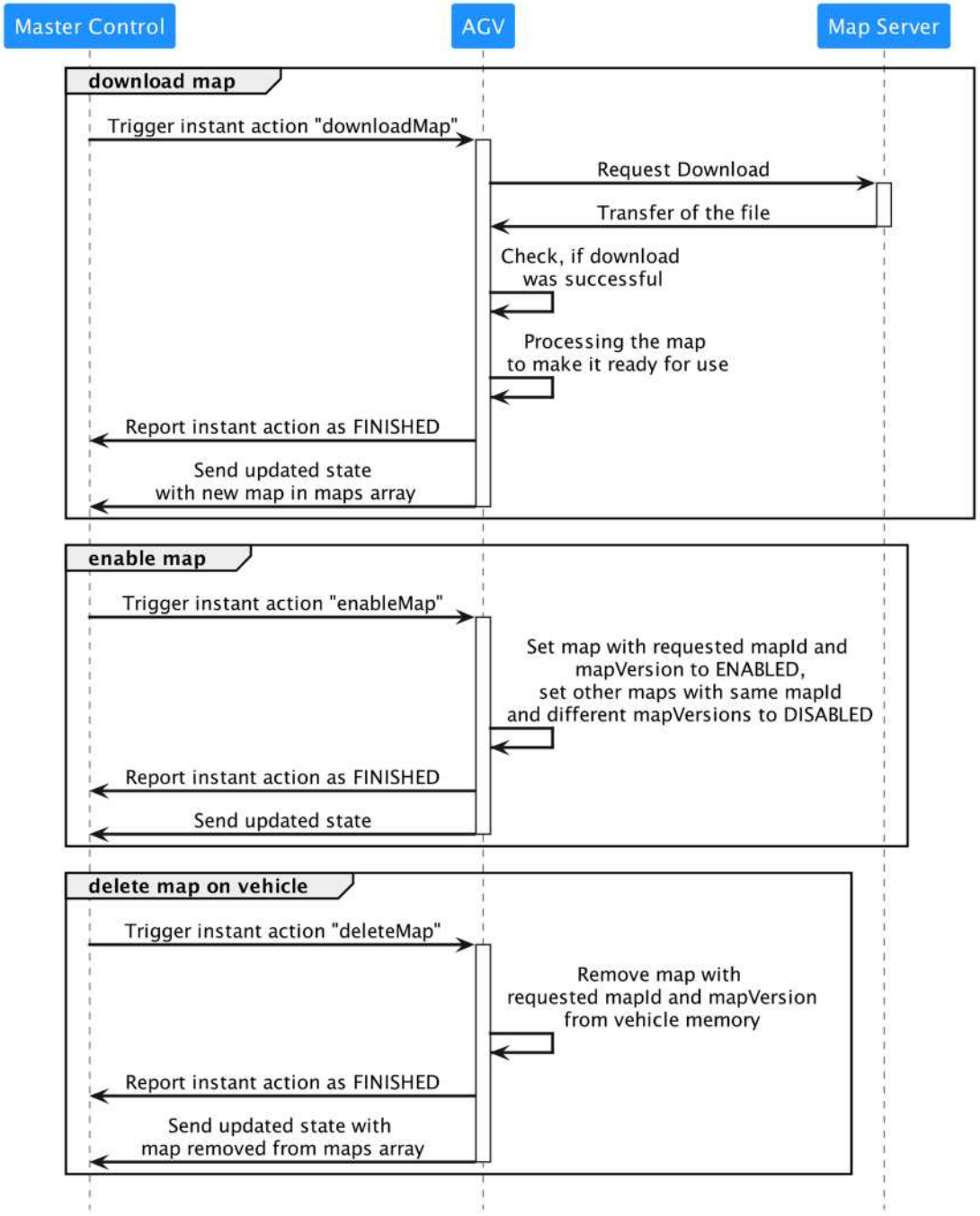


Figure 13 Communication required between master control, AGV and map server to download, enable, and delete a map.

6.7.2 Maps in the vehicle state

The `mapId` field in the `agvPosition` of the state represents the currently active map. Information about the maps available on a vehicle 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 vehicle 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 can only be one version of maps with the same `mapId` enabled at a time. If the `maps` array is empty, this means that there are currently no maps available on the vehicle.

6.7.3 Map download

The map download is triggered by the `downloadMap` instant action from the Master Control. This command contains the mandatory parameters `mapId` and `mapDownloadLink` under which the map is stored on the map server and which can be accessed by the vehicle.

The AGV 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, before they are ready to be enabled.

It is important to ensure that the process of downloading a map does not modify, delete, enable, or disable any existing maps on the vehicle. The vehicle shall reject the download of a map with a `mapId` and `mapVersion` that is already on the vehicle. An error shall be reported, and the status of the instant action shall be set to 'FAILED'. The master control shall first delete the map on the vehicle and then restart the download.

6.7.4 Enable downloaded maps

There are two ways to enable a map on a vehicle:

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

It is the responsibility of the master control to ensure that the correct maps are activated on the vehicle when sending the corresponding `mapId` as part of a `nodePosition` in an order. If the vehicle is to be set to a specific position on a new map, the `initPosition` instant action is used.

6.7.5 Delete maps on vehicle

The master control can request the deletion of a specific map from a vehicle. This is done with the instant action `deleteMap`. When a vehicle runs out of memory, it should report this to the master control, which can then initiate the deletion of maps. The vehicle itself is not allowed to delete maps. After successfully deleting a map, it is important to remove that map's entry from the vehicle's array of maps in the vehicle state.

6.8 Actions

If the AGV supports actions other than driving, these actions are executed via the action field that is attached to either a node or an edge, or sent via the separate topic instantActions (see Section 6.9 Topic: "instantActions" (from master to control to AGV).

Actions that are to be executed on an edge shall only run while the AGV is on the edge (see Section 6.10.2 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. Actions on nodes 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 should be formulated pairwise (e.g., activateWarningLights and deactivateWarningLights), although there may be exceptions.

The following section presents predefined actions that shall be used by the AGV, if the AGV'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.

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

6.8.1 Definition, parameters, effects and scope of predefined actions

general						scope		
action	**counter action**	**description**	**idempotent**	**parameters**	**linked state**	**instant**	**node**	**edge**
startPause	stopPause	Activates the pause mode. A linked state is required, because many AGVs can be paused by using a hardware switch. No more AGV driving movements - reaching next node is not necessary. Actions can continue. Order is resumable.	yes	-	paused	yes	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 AGVs can be paused by using a hardware switch. stopPause can also restart vehicles that were stopped with a hardware button that triggered startPause (if configured).	yes	-	paused	yes	no	no
startCharging	stopCharging	Activates the charging process. Charging can be done on a charging spot (vehicle standing) or on a charging lane (while driving). Protection against overcharging is responsibility of the vehicle.	yes	-	.batteryState.charging	yes	yes	no
stopCharging	startCharging	Deactivates the charging process to send a new order. The charging process can also be interrupted by the vehicle / charging station, e.g., if the battery is full. Battery state is only allowed to be "false", when AGV is ready to receive orders.	yes	-	.batteryState.charging	yes	yes	no

general						scope		
action	**counter action**	**description**	**idempotent**	**parameters**	**linked state**	**instant**	**node**	**edge**
initPosition	-	Resets (overrides) the pose of the AGV with the given parameters.	yes	x (float64) y (float64) theta (float64) mapId (string) lastNodeId (string)	.agvPosition.x .agvPosition.y .agvPosition.theta .agvPosition.mapId .lastNodeId .maps	yes	yes (Elevator)	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
downloadMap	-	Trigger the download of a new map. Active during the download. Errors reported in vehicle 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
deleteMap	-	Trigger the removal of a map from the vehicle memory.	yes	mapId (string) mapVersion (string)	.maps	yes	no	no
stateRequest	-	Requests the AGV to send a new state report.	yes	-	-	yes	no	no
logReport	-	Requests the AGV to generate and store a log report.	yes	reason (string)	-	yes	no	no
pick	drop (if automated)	Request the AGV to pick a load. AGVs 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).	no	lhd (string, optional) stationType (string) stationName(string, optional) loadType (string) loadId(string, optional) height (float64) (optional) defines bottom of the load related to the floor depth (float64) (optional) for	.load	no	yes	yes

general						scope		
action	**counter action**	**description**	**idempotent**	**parameters**	**linked state**	**instant**	**node**	**edge**
		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 AGV state, because the associated node is not released yet. If on an edge, the vehicle can use its sensing device to detect the position for picking the node.		forklifts side(string) (optional) e.g., conveyor				
drop	pick (if automated)	Request the AGV 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
detectObject	-	AGV detects object (e.g., load, charging spot, free parking position).	yes	objectType(string, optional)	-	no	yes	yes
finePositioning	-	On a node, AGV will position exactly on a target. The AGV is allowed to deviate from its node position. On an edge, AGV will e.g., align on stationary equipment while traversing an edge. InstantAction: AGV starts positioning exactly on a target.	yes	stationType(string, optional) stationName(string, optional)	-	no	yes	yes

general						scope		
action	**counter action**	**description**	**idempotent**	**parameters**	**linked state**	**instant**	**node**	**edge**
waitForTrigger	-	AGV has to wait for a trigger on the AGV (e.g., button press, manual loading). Master control is responsible to handle the timeout and has to cancel the order if necessary.	yes	triggerType(string)	-	no	yes	no
cancelOrder	-	AGV stops as soon as possible. This could be immediately or on the next node. Then the order is deleted. All actions are canceled.	yes	-	-	yes	no	no
factsheetRequest	-	Requests the AGV to send a factsheet	yes	-	-	yes	no	no

6.8.2 Predefined action definitions, description of their states

	action states				
action	**'INITIALIZING'**	**'RUNNING'**	**'PAUSED'**	**'FINISHED'**	**'FAILED'**
startPause	-	Activation of the mode is in preparation. If the AGV supports an instant transition, this state can be omitted.	-	Vehicle stands still. All actions will be paused. The pause mode is activated. The AGV 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 AGV supports an instant transition, this state can be omitted.	-	The pause mode is deactivated. All paused actions will be resumed. The AGV reports .paused: "false".	The pause mode cannot be deactivated for some reason (e.g., overwritten by hardware switch).
startCharging	-	Activation of the charging process is in	-	The charging process is started.	The charging process could not be started for

	action states				
action	**'INITIALIZING'**	**'RUNNING'**	**'PAUSED'**	**'FINISHED'**	**'FAILED'**
		progress (communication with charger is running). If the AGV supports an instant transition, this state can be omitted.		The AGV reports .batteryState.charging: "true".	some reason (e.g., not aligned to charger). Charging problems should correspond with an error.
stopCharging	-	Deactivation of the charging process is in progress (communication with charger is running). If the AGV supports an instant transition, this state can be omitted.	-	The charging process is stopped. The AGV reports .batteryState.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.
initPosition	-	Initializing of the new pose in progress (confidence checks, etc.). If the AGV supports an instant transition, this state can be omitted.	-	The pose is reset. The AGV reports .agvPosition.x = x, .agvPosition.y = y, .agvPosition.theta = theta .agvPosition.mapId = mapId .agvPosition.lastNodeId = lastNodeId	The pose is not valid or cannot be reset. General localization problems should correspond with an error.
downloadMap	Initialize the connection to the map server.	AGV is downloading the map, until download is finished.	-	AGV updates its state by setting the mapId/mapVersion and the corresponding mapStatus to 'DISABLED'.	The download failed, updated in vehicle state (e.g., connection lost, Map server unreachable, mapId/mapVersion not existing on map server).

	action states				
action	**'INITIALIZING'**	**'RUNNING'**	**'PAUSED'**	**'FINISHED'**	**'FAILED'**
enableMap	-	AGV enables the map with the requested mapId and mapVersion while disabling other versions with the same mapId.	-	The AGV updates the corresponding mapStatus of the requested map to 'ENABLED' and the other versions with same mapId to 'DISABLED''.	The requested mapId/mapVersion combination does not exist.
deleteMap	-	AGV deletes map with requested mapId and mapVersion from its internal memory.	-	AGV removes mapId/mapVersion from its state.	Can not delete map, if map is currently in use. The requested mapId/mapVersion combination was already deleted before.
stateRequest	-	-	-	The state has been communicated	-
logReport	-	The report is in generating. If the AGV supports an instant generation, this state can be omitted.	-	The report is stored. The name of the log will be reported in status.	The report can not be stored (e.g., no space).
pick	Initializing of the pick process, e.g., outstanding lift operations.	The pick process is running (AGV 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 is done. Load has entered the AGV and AGV reports new load state.	Pick failed, e.g., station is unexpected empty. Failed pick operations should correspond with an error.

	action states				
action	**'INITIALIZING'**	**'RUNNING'**	**'PAUSED'**	**'FINISHED'**	**'FAILED'**
drop	Initializing of the drop process, e.g., outstanding lift operations.	The drop process is running (AGV 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 is done. Load has left the AGV and AGV reports new load state.	Drop failed, e.g., station is unexpected occupied. Failed drop operations should correspond with an error.
detectObject	-	Object detection is running.	-	Object has been detected.	AGV could not detect the object.
finePositioning	-	AGV positions itself exactly on a target.	The fine positioning process is being paused, e.g., if a safety field is violated. After removing the violation, the fine positioning continues.	Goal position in reference to the station is reached.	Goal position in reference to the station could not be reached.
waitForTrigger	-	AGV is waiting for the trigger	-	Trigger has been triggered.	waitForTrigger fails, if order has been canceled.
cancelOrder	-	AGV is stopping or driving, until it reaches the next node.	-	AGV stands still and has canceled the order.	-
factsheetRequest	-	-	-	The factsheet has been communicated	-

6.9 Topic: "instantActions" (from master to control to AGV)

In certain cases, it is necessary to send actions to the AGV that need to be performed immediately. This is made possible by publishing an instantAction message to the topic instantActions. instantActions shall not conflict with the content of the AGV'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 AGV without changing anything in the current order;
- resume order after pause;
- activate signal (optical, audio, etc.).

For additional information, see Section 7 Best practice.

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.ffZ (e.g., "2017-04-15T11:40:03.12Z")
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the AGV.
serialNumber	string	Serial number of the AGV.
actions [action]	array	Array of actions that need to be performed immediately and are not part of the regular order.

When an AGV receives an instantAction, an appropriate actionStatus is added to the actionStates array of the AGV state. The actionStatus is updated according to the progress of the action. See also Figure 16 for the different transitions of an actionStatus.

6.10 Topic: "state" (from AGV to master control)

The AGV state will be transmitted on only one topic. Compared to separate messages (e.g., for orders, battery state and errors) using one topic will reduce the workload of the broker and the master control for handling messages, while also keeping the information about the AGV state synchronized.

AGV state message will be published with occurrence of relevant events or at the latest every 30s via MQTT broker to master control.

Events that trigger the transmission of the state message are:

- Receiving an order
- Receiving an order update
- Changes in the load status
- Errors or warnings
- Driving over a node
- Switching the operating mode
- Change in the `driving` field
- Change in the `nodeStates`, `edgeStates`, actionStates or maps

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 node- and edgeStates; as does the driving over a node), it is sensible to trigger one state update instead of multiple.

6.10.1 Concept and Logic

The order progress is tracked by the `nodeStates` and `edgeStates`. Additionally, if the AGV is able to derive its current position, it can publish its position via the `position` field.

If the AGV plans the path by itself, it shall communicate its calculated trajectory (including base and horizon) in the form of NURBS via the `trajectory` object in the state message, unless master control cannot use this field, and it was agreed during integration, that this field shall not be sent. After nodes are released by master control, the AGV is not allowed to change its trajectory.

The `nodeStates` and `edgeStates` includes all nodes/edges, that the AGV still shall traverse.

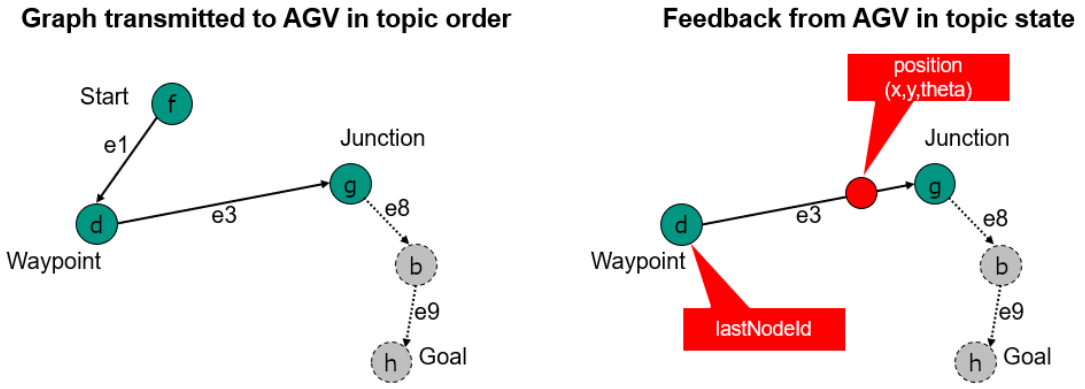


Figure 14 Order information provided by the state topic. Only the ID of the last node and the remaining nodes and edges are transmitted

6.10.2 Traversal of nodes and entering/leaving edges, triggering of actions

The AGV decides on its own, when a node should count as traversed. Generally, the AGV's control point should be within the node's `allowedDeviationXY` and its orientation within `allowedDeviationTheta`. If the edge attribute `corridor` of the subsequent edge is set, these boundaries should be met additionally.

The AGV reports the traversal of a node by removing its `nodeState` from the `nodeStates` array and setting the `lastNodeId`, `lastNodeSequenceId` to the traversed node's values.

As soon as the AGV reports the node as traversed, the AGV shall trigger the actions associated with the node, if any.

The traversal of a node also marks the leaving of the edge leading up to the node. The edge shall then 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 AGV enters the following edge, if there is one. The edge's actions shall now be triggered. An exception to this rule is, if the AGV has to pause on the edge (because of a soft or hard blocking edge, or otherwise) – then the AGV enters the edge after it begins moving again.

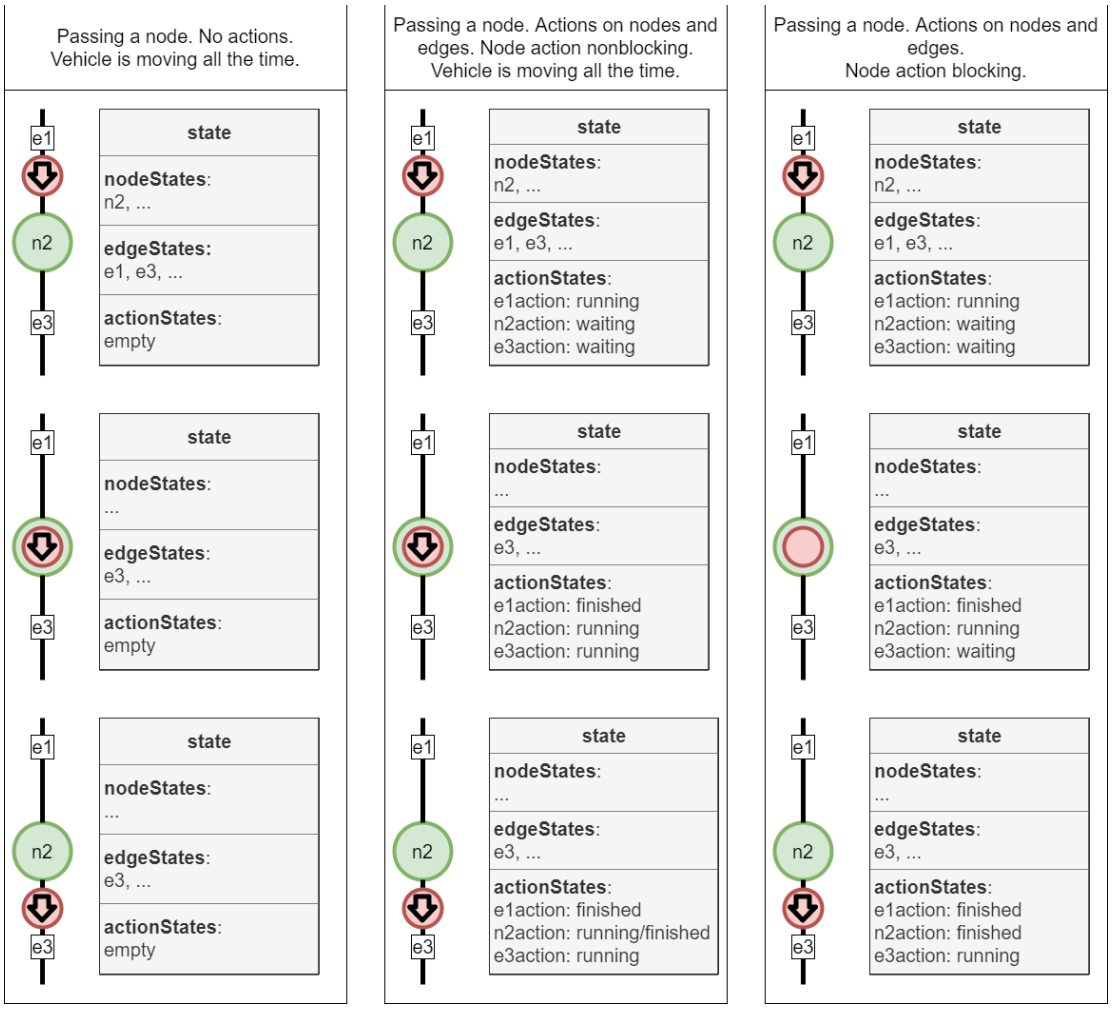


Figure 15 Depiction of nodeStates, edgeStates, actionStates during order handling

6.10.3 Base request

If the AGV detects that its base is running low, it can set the `newBaseRequest` flag to "true" to prevent unnecessary braking.

6.10.4 Information

The AGV can submit arbitrary additional information to master control via the `information` array. It is up to the AGV how long it reports information via an information message.

Master control shall not use the info messages for logic, it shall only be used for visualization and debugging purposes.

6.10.5 Errors

The AGV reports errors via the `errors` array. Errors have two levels: 'WARNING' and 'FATAL'. A 'WARNING' is a self-resolving error, e.g., a field violation. A 'FATAL' error needs human intervention. Errors can pass references that help with finding the cause of the error via the `errorReferences` array.

6.10.6 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.ffZ (e.g., "2017-04-15T11:40:03.12Z").
version		string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer		string	Manufacturer of the AGV.
serialNumber		string	Serial number of the AGV.
maps[map]		array	Array of map objects that are currently stored on the vehicle.
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 AGV. "0" if no previous orderUpdateId is available.
zoneSetId		string	Unique ID of the zone set, that the AGV currently uses for path planning. Shall be the same as the one used in the order. Optional: If the AGV does not use zones, this field can be omitted.
lastNodeId		string	Node ID of last reached node or, if AGV 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 AGV is currently on a node, Sequence ID of current node. "0" if no lastNodeSequenceId is available.
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)

Object structure	Unit	Data type	Description
agvPosition		JSON object	Current position of the AGV on the map. Optional: Can only be omitted for AGV without the capability to localize themselves, e.g., line-guided AGVs.
velocity		JSON object	The AGV velocity in vehicle coordinates.
loads [load]		array	Loads, that are currently handled by the AGV. Optional: If the AGV cannot determine the load state, this field shall be omitted completely and not be reported as an empty array. If the AGV can determine the load state, but the array is empty, the AGV is considered unloaded.
driving		boolean	"true": indicates, that the AGV is driving and/or rotating. Other movements of the AGV (e.g., lift movements) are not included here. "false": indicates that the AGV is neither driving nor rotating.
paused		boolean	"true": AGV is currently in a paused state, either because of the push of a physical button on the AGV or because of an instantAction. The AGV can resume the order. "false": The AGV is currently not in a paused state.
newBaseRequest		boolean	"true": AGV is almost at the end of the base and will reduce speed, if no new base is transmitted. Trigger for master control to send a new base. "false": no base update required.
distanceSinceLastNode	meter	float64	Used by line-guided vehicles to indicate the distance it has been driving past the "lastNodeId". Distance is in meters.

Object structure	Unit	Data type	Description
actionStates [actionState]		array	Contains an array of all actions from the current order and all received instantActions since the last order. The action states are kept until a new order is received. Action states, except for running instant actions, are removed upon receiving 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`.
batteryState		JSON object	Contains all battery-related information.
operatingMode		string	Enum {'AUTOMATIC', 'SEMIAUTOMATIC', 'MANUAL', 'SERVICE', 'TEACHIN'} For additional information, see Table 1 in Section 6.10.6 Implementation of the state message
errors [error]		array	Array of error objects. All active errors of the AGV should be in the array. An empty array indicates that the AGV has no active errors.
information [info]		array	Array of info objects. An empty array indicates, that the AGV has no information. This should only be used for visualization or debugging – it shall not be used for logic in master 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 vehicle's workspace.
mapVersion		string	Version of the map.
mapDescription		string	Additional information on the map.

Object structure	Unit	Data type	Description
mapStatus }		string	Enum {'ENABLED', 'DISABLED'} 'ENABLED': Indicates this map is currently active / used on the AGV. 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 AGV and thus could be enabled or deleted by request.

Object structure	Unit	Data type	Description
nodeState {		JSON object	
nodeId		string	Unique node identification.
sequenceId		uint32	sequence ID to discern multiple nodes with same nodeId.
nodeDescription		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. The object is defined in 6.6 Topic: "order" (from master control to AGV) Optional: Master control has this information. Can be sent additionally, e.g., for debugging purposes.

Object structure	Unit	Data type	Description
edgeState {		JSON object	
edgeId		string	Unique edge identification.
sequenceId		uint32	sequence ID to differentiate between multiple edges with the same edgeId.
edgeDescription		string	Additional information on the edge.
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	The trajectory is to be communicated as NURBS and is defined in Section 6.6.6 Implementation of the order message Trajectory segments start from the point, where the vehicle enters the edge, and terminate at the point, where the vehicle reports that the end node was traversed.

Object structure	Unit	Data type	Description
agvPosition {		JSON object	Defines the position on a map in world coordinates. Each floor has its own map.
positionInitialized		boolean	“true”: position is initialized. “false”: position is not initialized.
localizationScore		float64	Range: [0.0 ... 1.0] Describes the quality of the localization and therefore, can be used, e.g., by SLAM AGV to describe, how accurate the current position information is. 0.0: position unknown 1.0: position known Optional for vehicles that cannot estimate their localization score. Only for logging and visualization purposes.
deviationRange	m	float64	Value for the deviation range of the position in meters. Optional for vehicles that cannot estimate their deviation e.g., grid-based localization. Only for logging and visualization purposes.
x	m	float64	X-position on the map in reference to the map coordinate system. Precision is up to the specific implementation.
y	m	float64	Y-position on the map in reference to the map coordinate system. Precision is up to the specific implementation.
theta		float64	Range: [-Pi ... Pi] Orientation of the AGV.
mapId		string	Unique identification of the map in which the position is referenced. Each map has the same origin of coordinates. When an AGV 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.
mapDescription }		string	Additional information on the map.

Object structure	Unit	Data type	Description
velocity {		JSON object	
vx	m/s	float64	The AGV's velocity in its X-direction.

Object structure	Unit	Data type	Description
vy	m/s	float64	The AGV's velocity in its Y-direction.
omega }	Rad/s	float64	The AGV'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 AGV can identify the load but didn't identify the load yet. Optional if the AGV cannot identify the load.
loadType		string	Type of load.
loadPosition		string	Indicates, which load handling/carrying unit of the AGV is used, e.g., in case the AGV has multiple spots/positions to carry loads. For example: "front", "back", "positionC1", etc. Optional for vehicles 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 AGV'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 AGV's coordinate system.
x		float64	X-coordinate of the point of reference.
y		float64	Y-coordinate of the point of reference.

Object structure	Unit	Data type	Description
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 of the load's bounding box.
width	m	float64	Absolute width 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	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. MC is aware of action type as dispatched in the order.
actionDescription		string	Additional information on the current action.
actionStatus		string	Enum {'WAITING', 'INITIALIZING', 'RUNNING', 'PAUSED', 'FINISHED', 'FAILED'} See Section 6.11 Action states.
resultDescription }		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
batteryState {		JSON object	
batteryCharge	%	float64	State of Charge: if AGV only provides values for good or bad battery levels, these will be indicated as 20% (bad) and 80% (good).

Object structure	Unit	Data type	Description
batteryVoltage	V	float64	Battery Voltage.
batteryHealth	%	int8	Range: [0 ... 100] State describing the battery's health.
charging		boolean	“true”: charging in progress. “false”: AGV is currently not charging.
reach }	m	uint32	Range: [0 ... uint32.max] Estimated reach with current state of charge.

Object structure	Unit	Data type	Description
error {		JSON object	
errorType		string	Type/name of error
errorReferences [errorReference]		array	Array of references (e.g., nodeId, edgeId, orderId, actionId, etc.) to provide more information related to the error. For additional information see 7 Best practice.
errorDescription		string	Verbose description providing details and possible causes of the error.
errorHint		string	Hint on how to approach or solve the reported error.
errorLevel }		string	Enum {'WARNING', 'FATAL'} 'WARNING': AGV is ready to start (e.g., maintenance cycle expiration warning). 'FATAL': AGV is not in running condition, user intervention required (e.g., laser scanner is contaminated).

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
info {		JSON object	
infoType		string	Type/name of information.
infoReferences [infoReference]		array	Array of references.

Object structure	Unit	Data type	Description
infoDescription		string	Description of the information.
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	
eStop		string	Enum {'AUTOACK', 'MANUAL', 'REMOTE', 'NONE'} Acknowledge-Type of eStop: 'AUTOACK': auto-acknowledgeable e-stop is activated, e.g., by bumper or protective field. 'MANUAL': e-stop hast to be acknowledged manually at the vehicle. 'REMOTE': facility e-stop has to be acknowledged remotely. 'NONE': no e-stop activated.
fieldViolation }		boolean	Protective field violation. "true":field is violated "false":field is not violated.

Operating Mode Description

The following description lists the operatingMode of the topic "state".

Identifier	Description
AUTOMATIC	AGV is under full control of the master control. AGV drives and executes actions based on orders from the master control.
SEMIAUTOMATIC	AGV is under control of the master control. AGV drives and executes actions based on orders from the master control. The driving speed is controlled by the HMI (speed can't exceed the speed of automatic mode). The steering is under automatic control (non-safe HMI possible).
MANUAL	Master control is not in control of the AGV. Supervisor doesn't send driving order or actions to the AGV. HMI can be used to control the steering and velocity and handling device of the AGV. Location of the AGV is sent to the master control. When AGV enters or leaves this mode, it immediately clears all the orders (safe HMI required).
SERVICE	Master control is not in control of the AGV. Master control doesn't send driving order or actions to the AGV. Authorized personnel can reconfigure the AGV.
TEACHIN	Master control is not in control of the AGV. Supervisor doesn't send driving order or actions to the AGV. The AGV is being taught, e.g., mapping is done by a master control.

Table 1 The operating modes and their meaning

6.11 Action states

When an AGV receives an `action` (either attached to a `node` or `edge` or via an `instantAction`), it shall represent this `action` with an `actionState` in its `actionStates` array.

`actionStates` describe in the field `actionStatus` at which stage of the action's life cycle the action is.

Table 2 describes, which value the enum `actionStatus` can hold.

actionStatus	Description
'WAITING'	Action was received by AGV but the node where it triggers was not yet reached or the edge where it is active 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 AGV)
'FINISHED'	The action is finished. A result is reported via the resultDescription.
'FAILED'	Action could not be finished for whatever reason.

Table 2 The acceptable values for the actionStatus field

A state transition diagram is provided in Figure 16.

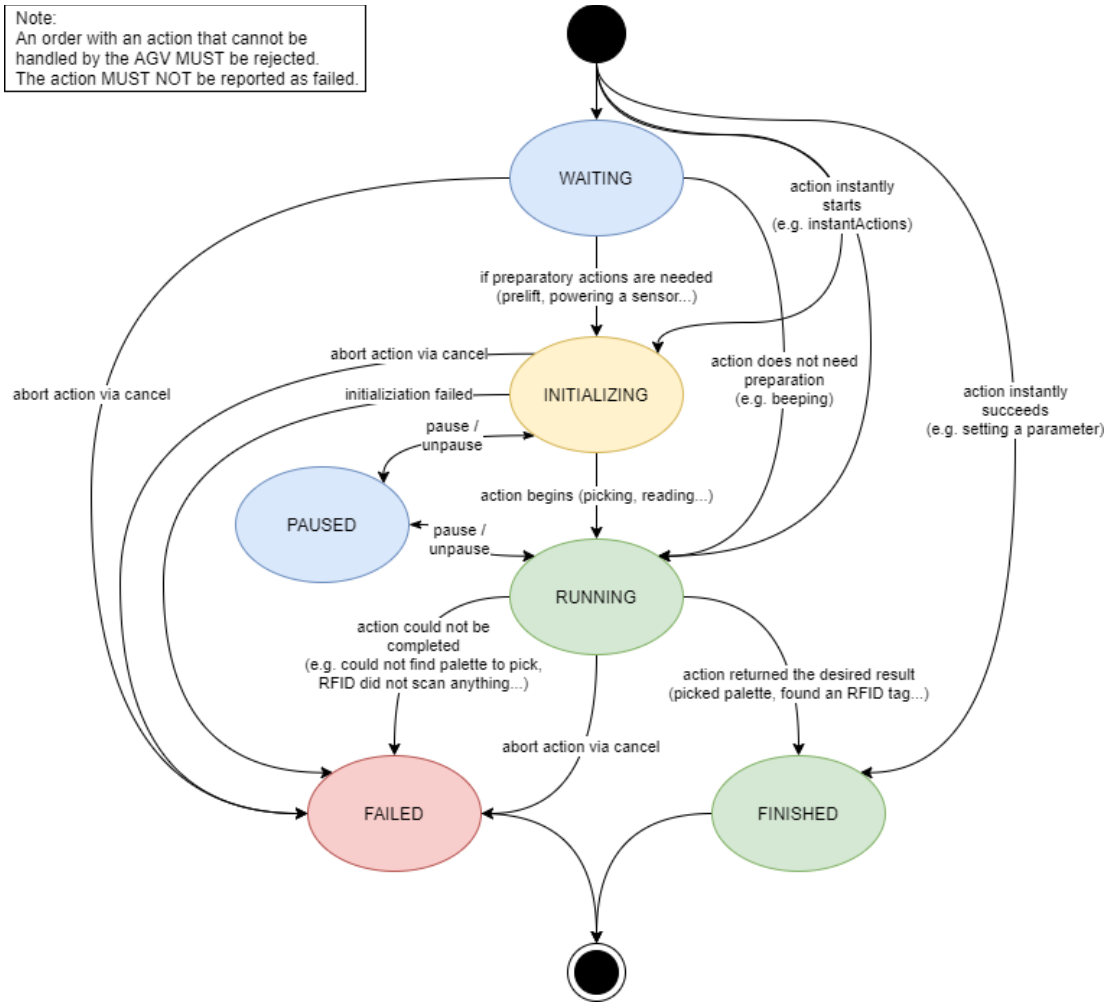


Figure 16 All possible status transitions for actionStates

6.12 Action blocking types and sequence

The order of multiple actions in a list define the sequence, in which those actions are to be executed. The parallel execution of actions is governed by their respective `blockingType`.

Actions can have three distinct blocking types, described in Table 3.

actionStatus	Description
NONE	Action can be executed in parallel with other actions and while the vehicle is driving.
SOFT	Action can be executed in parallel with other actions. Vehicle shall not drive.
HARD	Action shall not be executed in parallel with other actions. Vehicle shall not drive.

Table 3 Action blocking types

If there are multiple actions on the same node with different blocking types, Figure 17 describes how the AGV should handle these actions.

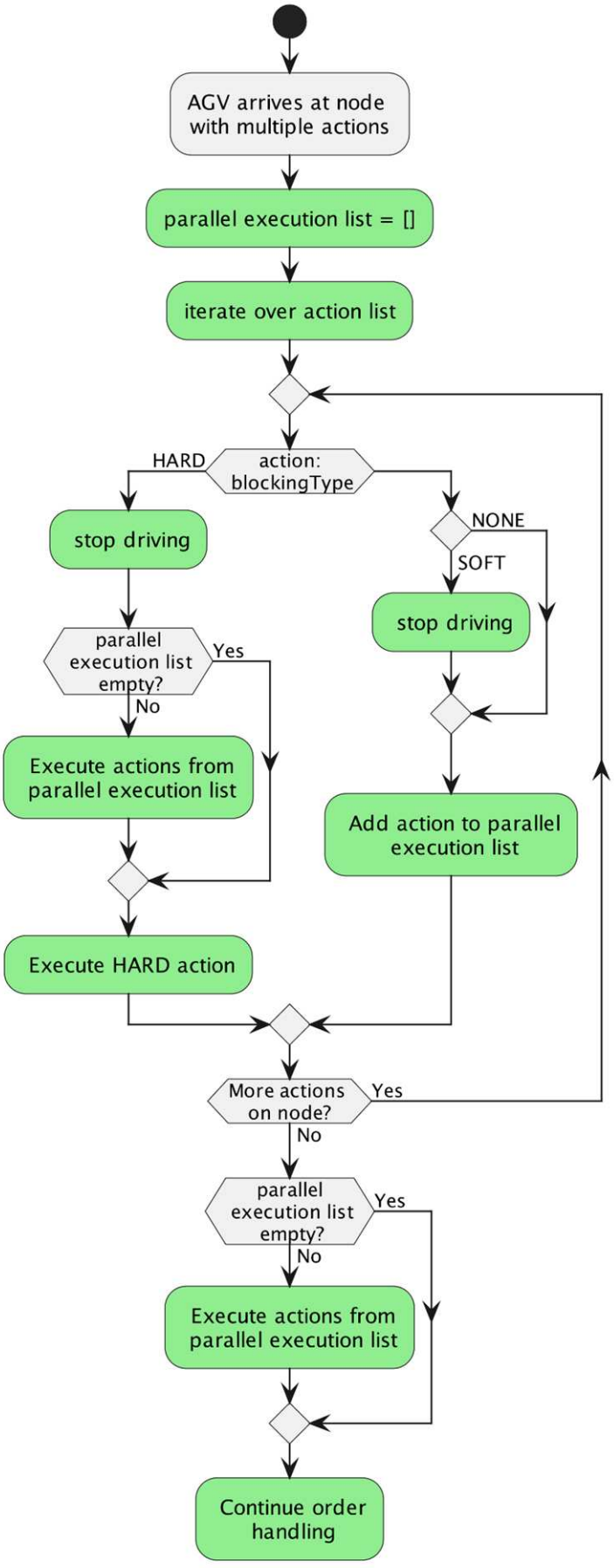


Figure 17 Handling multiple actions

6.13 Topic "visualization"

For a near real-time position update the AGV can broadcast its position and velocity on the topic `visualization`.

The structure of the position object is the same as the position and velocity object in the state. For additional information see Section 6.10.6 Implementation of the state message for the vehicle state. The update rate for this topic is defined by the integrator.

6.14 Topic "connection"

During the connection of an AGV client to the broker, a last will topic and message can be set, which is published by the broker upon disconnection of the AGV client from the broker. Thus, the master control can detect a disconnection event by subscribing the connection topics of all AGV. The disconnection is detected via a heartbeat that is exchanged between the broker and the client. The interval is configurable in most brokers and should be set around 15 seconds. The Quality of Service level for the `connection` topic shall be 1 - At Least Once.

The suggested last will topic structure is:

`uagv/v2/manufacturer/SN/connection`

The last will message is defined as a JSON encapsulated message with the following fields:

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 (ISO8601, UTC); YYYY-MM-DDTHH:mm:ss.ffZ(e.g., "2017-04-15T11:40:03.12Z").
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the AGV.
serialNumber	string	Serial number of the AGV.
connectionState	string	Enum {'ONLINE', 'OFFLINE', 'CONNECTIONBROKEN'} 'ONLINE': connection between AGV and broker is active. 'OFFLINE': connection between AGV and broker has gone offline in a coordinated way. 'CONNECTIONBROKEN': The connection between AGV and broker has unexpectedly ended.

The last will message will not be sent, when a connection is ended in a graceful way by using an MQTT disconnection command. The last will message is only sent by the broker, if the connection is unexpectedly interrupted.

Note: Due to the nature of the last will feature in MQTT, the last will message is defined during the connection phase between the AGV and the MQTT broker. As a result, the timestamp and headerId fields will always be outdated.

AGV wants to disconnect gracefully:

1. AGV sends "uagv/v2/manufacturer/SN/connection" with `connectionState` set to `OFFLINE`.
2. Disconnect the MQTT connection with a disconnect command.

AGV comes online:

1. Set the last will to "uagv/v2/manufacturer/SN/connection" with the field `connectionState` set to `CONNECTIONBROKEN`, when the MQTT connection is created.
2. Send the topic "uagv/v2/manufacturer/SN/connection" with `connectionState` set to `ONLINE`.

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

When connection between the AGV and the broker stops unexpectedly, the broker will send the last will topic: "uagv/v2/manufacturer/SN/connection" with the field `connectionState` set to `CONNECTIONBROKEN`.

6.15 Topic "factsheet"

The factsheet provides basic information about a specific AGV type series. This information allows comparison of different AGV types and can be applied for the planning, dimensioning, and simulation of an AGV system. The factsheet also includes information about AGV communication interfaces which are required for the integration of an AGV type series into a VDA-5050-compliant master control.

The values for some fields in the AGV factsheet can only be specified during system integration, for example the assignment of project-specific load and station types, together with the list of station and load types which are supported by this AGV.

The factsheet is intended as both a human-readable document and for machine processing, e.g., an import by the master control application, and thus is specified as a JSON document.

The MC can request the factsheet from the AGV by sending the instant action: `factsheetRequest`

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

6.15.1 Factsheet JSON structure

The factsheet consists of the JSON objects listed in the following table.

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 (ISO8601, UTC); YYYY-MM-DDTHH:mm:ss.ffZ(e.g., "2017-04-15T11:40:03.12Z").
version	string	Version of the protocol [Major].[Minor].[Patch] (e.g., 1.3.2).
manufacturer	string	Manufacturer of the AGV.
serialNumber	string	Serial number of the AGV.
typeSpecification	JSON object	These parameters generally specify the class and the capabilities of the AGV.
physicalParameters	JSON object	These parameters specify the basic physical properties of the AGV.
protocolLimits	JSON object	Limits for length of identifiers, arrays, strings, and similar in MQTT communication.
protocolFeatures	JSON object	Supported features of VDA5050 protocol.
agvGeometry	JSON object	Detailed definition of AGV geometry.

Field	data type	description
loadSpecification	JSON object	Abstract specification of load capabilities.
vehicleConfig	JSON object	Summary of current software and hardware versions on the vehicle and optional network information.

typeSpecification

This JSON object describes general properties of the AGV 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 AGV type series.
agvKinematic	string	Simplified description of AGV kinematics type. [DIFF, OMNI, THREEWHEEL] DIFF: differential drive OMNI: omni-directional vehicle THREEWHEEL: three-wheel-driven vehicle or vehicle with similar kinematics
agvClass	string	Simplified description of AGV class. [FORKLIFT, CONVEYOR, TUGGER, CARRIER] FORKLIFT: forklift. CONVEYOR: AGV with conveyors on it. TUGGER: tugger. CARRIER: load carrier with or without lifting unit.
maxLoadMass	float64	[kg], Maximum loadable mass.
localizationTypes	array of string	Simplified description of localization type. Example values: 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 AGV, sorted by priority. Example values: PHYSICAL_LINE_GUIDED: No path planning, AGV follows physical installed paths. VIRTUAL_LINE_GUIDED: AGV goes fixed (virtual) paths. AUTONOMOUS: AGV plans its path autonomously.

physicalParameters

This JSON object describes physical properties of the AGV.

Field	data type	description
speedMin	float64	[m/s] Minimal controlled continuous speed of the AGV.
speedMax	float64	[m/s] Maximum speed of the AGV.
angularSpeedMin	float64	[Rad/s] Minimal controlled continuous rotation speed of the AGV.
angularSpeedMax	float64	[Rad/s] Maximum rotation speed of the AGV.
accelerationMax	float64	[m/s²] Maximum acceleration with maximum load.
decelerationMax	float64	[m/s²] Maximum deceleration with maximum load.
heightMin	float64	[m] Minimum height of AGV.
heightMax	float64	[m] Maximum height of AGV.
width	float64	[m] Width of AGV.
length	float64	[m] Length of AGV.

protocolLimits

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

Field	data type	description
maxStringLens {	JSON object	Maximum lengths of strings.
msgLen	uint32	Maximum MQTT message length.
topicSerialLen	uint32	Maximum length of serial number part in MQTT-topics.Affected parameters:order.serialNumberinstantActions.serialNumberstate.SerialNumbervisualization.serialNumberconnection.serialNumber
topicElemLen	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 visualization.manufacturer connection.timestamp connection.version connection.manufacturer

Field	data type	description
idLen	uint32	Maximum length of ID strings. Affected parameters: order.orderId order.zoneSetId node.nodeId nodePosition.mapId action.actionId edge.edgeId edge.startNodeId edge.endNodeId
idNumericalOnly	boolean	If "true" ID strings need to contain numerical values only.
enumLen	uint32	Maximum length of enum and key strings. Affected parameters: action.actionType action.blockingType edge.direction actionParameter.key state.operatingMode load.loadPosition load.loadType actionState.actionStatus error.errorType error.errorLevel errorReference.referenceKey info.infoType info.infoLevel safetyState.eStop connection.connectionState
loadIdLen	uint32	Maximum length of loadId strings.
}		
maxArrayLens {	JSON object	Maximum lengths of arrays.
order.nodes	uint32	Maximum number of nodes per order processable by the AGV.
order.edges	uint32	Maximum number of edges per order processable by the AGV.
node.actions	uint32	Maximum number of actions per node processable by the AGV.
edge.actions	uint32	Maximum number of actions per edge processable by the AGV.
actions.actionsParameters	uint32	Maximum number of parameters per action processable by the AGV.
instantActions	uint32	Maximum number of instant actions per message processable by the AGV.
trajectory.knotVector	uint32	Maximum number of knots per trajectory processable by the AGV.
trajectory.controlPoints	uint32	Maximum number of control points per trajectory processable by the AGV.
state.nodeStates	uint32	Maximum number of nodeStates sent by the AGV, maximum number of nodes in base of AGV.

Field	data type	description
state.edgeStates	uint32	Maximum number of edgeStates sent by the AGV, maximum number of edges in base of AGV.
state.loads	uint32	Maximum number of load objects sent by the AGV.
state.actionStates	uint32	Maximum number of actionStates sent by the AGV.
state.errors	uint32	Maximum number of errors sent by the AGV in one state message.
state.information	uint32	Maximum number of information sent by the AGV in one state message.
error.errorReferences	uint32	Maximum number of error references sent by the AGV for each error.
information.infoReferences	uint32	Maximum number of info references sent by the AGV for each information.
}		
timing {	JSON object	Timing information.
minOrderInterval	float32	[s], Minimum interval sending order messages to the AGV.
minStateInterval	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 actions and parameters which are supported by the AGV.

Field	data type	description
optionalParameters [optionalParameter]	array of JSON object	Array of supported and/or required optional parameters. Optional parameters that are not listed here are assumed to be not supported by the AGV.
{		
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 AGV operation.

Field	data type	description
description	string	Free-form text: description of optional parameter, e.g., - Reason, why the optional parameter `direction` is necessary for this AGV type and which values it can contain. - The parameter `nodeMarker` shall contain unsigned integers only. - NURBS support is limited to straight lines and circle segments.
}		
agvActions [agvAction]	array of JSON object	Array of all actions with parameters supported by this AGV. 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. For example: ['INSTANT', 'NODE']
actionParameters [actionParameter]	array of JSON object	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 6.6.6 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', 'FLOAT', 'STRING', 'OBJECT', 'ARRAY'.
description	string	Free-form text: description of the parameter.
isOptional	boolean	"true": optional parameter.
}		
resultDescription	string	Free-form text: description of the result.
blockingTypes	array of enum	Array of possible blocking types for defined action. Enum {'NONE', 'SOFT', 'HARD'}
}		

agvGeometry

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

Field	data type	description
wheelDefinitions [wheelDefinition]	array of JSON object	Array of wheels, containing wheel arrangement and geometry.
{		
type	enum	Wheel type 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 AGV coordinate. system
y	float64	[m], y-position in AGV coordinate. system
theta	float64	[rad], orientation of wheel in AGV 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 of JSON object	Array of AGV envelope curves in 2D, e.g., the mechanical envelopes for unloaded and loaded state, the safety fields for different speed cases.
{		
set	string	Name of the envelope curve set.
polygonPoints [polygonPoint]	array of JSON object	Envelope curve as an x/y-polygon polygon is assumed as closed and shall be non-self-intersecting.
{		
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.
}		

Field	data type	description
envelopes3d [envelope3d]	array of JSON object	Array of AGV envelope curves in 3D.
{		
set	string	Name of the envelope curve set.
format	string	Format of data, e.g., DXF.
data	JSON object	3D-envelope curve data, format specified in 'format'.
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 AGV.

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 doesn't exist or is empty, the AGV has no load handling device.*
loadSets [loadSet]	array of JSON object	Array of load sets that can be handled by the AGV
{		
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 AGV.*
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.
maxWeight	float64	[kg], maximum weight of load type.
minLoadhandlingHeight	float64	[m], minimum allowed height for handling of this load type and weight references to boundingBoxReference.

Field	data type	description
maxLoadhandlingHeight	float64	[m], maximum allowed height for handling of this load type and weight references to boundingBoxReference.
minLoadhandlingDepth	float64	[m], minimum allowed depth for this load type and weight references to boundingBoxReference.
maxLoadhandlingDepth	float64	[m], maximum allowed depth for this load type and weight references to boundingBoxReference.
minLoadhandlingTilt	float64	[rad], minimum allowed tilt for this load type and weight.
maxLoadhandlingTilt	float64	[rad], maximum allowed tilt for this load type and weight.
agvSpeedLimit	float64	[m/s], maximum allowed speed for this load type and weight.
agvAccelerationLimit	float64	[m/s²], maximum allowed acceleration for this load type and weight.
agvDecelerationLimit	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.
}		

vehicleConfig

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

Field	data type	description
versions[versionInfo]	array of JSON object	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 vehicle's network connection. The listed information shall not be updated while the vehicle is operating.
dnsServers	array of string	Array of Domain Name Servers (DNS) used by the vehicle.
ntpServers	array of string	Array of Network Time Protocol (NTP) servers used by the vehicle.

Field	data type	description
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 vehicle, corresponding to the local IP address.
}		

7 Best practice

This section includes additional information, which helps in facilitating a common understanding concurrent with the logic of the protocol.

7.1 Error reference

If an error occurs due to an erroneous order, the AGV should return a meaningful error reference in the field errorReferences (see Section 6.10.6 Implementation of the state messageof the state topic). 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

If an action cannot be completed because of external factors (e.g., no load at expected position), the actionId should be referenced.

7.2 Format of 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, 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 float, bool, and even an object.

8 Glossary

8.1 Definition

Concept	Description
Free navigation AGV	Vehicles that use a map to plan their own path. The master control sends only start and destination coordinates. The vehicle sends its path to the master control. When connection to the master control is broken, the vehicle is able to continue its journey. Free-navigation vehicles may be allowed to bypass local obstacles. It may also be possible that a fine adjustment of the receiving/dispensing position are made by the vehicle itself.
Guided vehicles (physical or virtual)	Vehicles that get their path sent by the master control. The calculation of the path takes place in the master control. When communication to the master control is broken off, the vehicle terminates its released nodes and edges (the "base") and then stops. Guided vehicles may be allowed to bypass local obstacles. It may also be possible that fine adjustments of the receiving/dispensing position are made by the vehicle itself.
Central map	The maps that will be held centrally in the master control. This is initially created and then used.

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 2.1.0, January 2025