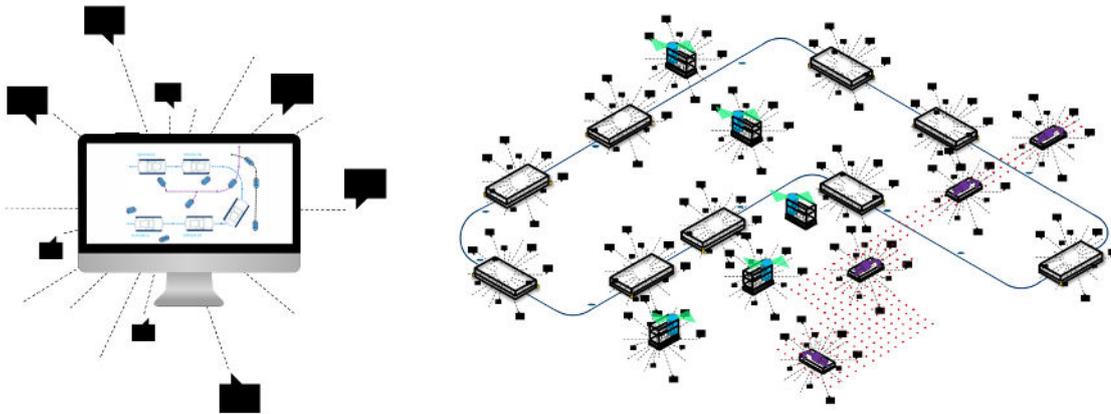


## VDA Recommendation

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

VDA 5050

Version 2.0.0, January 2022



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 (AGV) for intralogistics processes.

## Disclaimer

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

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

# Table of contents

<b>1</b>	<b>Foreword</b>	<b>5</b>
<b>2</b>	<b>Objective of the document</b>	<b>5</b>
<b>3</b>	<b>Scope</b>	<b>7</b>
	3.1 Other applicable documents	7
<b>4</b>	<b>Requirements and protocol definition</b>	<b>8</b>
<b>5</b>	<b>Process and content of communication</b>	<b>9</b>
<b>6</b>	<b>Protocol specification</b>	<b>11</b>
	6.1 Symbols of the tables and meaning of formatting	11
	6.2 MQTT connection handling, security and QoS	12
	6.3 MQTT-Topic Levels	12
	6.4 Protocol Header	13
	6.5 Subtopics for communication	13
	6.6 Topic: "order"(from master control to AGV)	14
	6.7 Implementation of the order message	22
	6.8 Actions	28
	6.9 Topic: "instantActions" (from master to control to AGV)	34
	6.10 Topic: "state" (from AGV to master control)	34
	6.11 actionStates	45
	6.12 Action Blocking Types and Sequence	46
	6.13 Topic "visualization"	48
	6.14 Topic "connection"	48
	6.15 Topic "factsheet"	49
<b>7</b>	<b>Best practice</b>	<b>58</b>
	7.1 Error reference	58
	7.2 Format of parameters	58
<b>8</b>	<b>Glossary</b>	<b>59</b>

## List of Figures

Figure 1	Integration of DTS inventory systems .....	6
Figure 2	Structure of the Information Flow .....	9
Figure 3	Graph representation in Master Control and graph transmitted in orders .....	14
Figure 4	Procedure for changing the driving route "Horizon" .....	15
Figure 5	Pseudocode of an order .....	16
Figure 6	Pseudocode of an order update. Please look out for the change of the "orderUpdateId" .....	16
Figure 7	Regular update process - order extension .....	17
Figure 8	The process of accepting an order or orderUpdate .....	18
Figure 9	Expected behavior after a cancelOrder .....	19
Figure 10	Coordinate system with sample AGV and orientation .....	21
Figure 11	Coordinate systems for map and vehicle .....	22
Figure 12	Order Information provided by the state topic. Only the ID of the last node and the remaining nodes and edges are transmitted .....	35
Figure 13	nodeStates, edgeStates, actionStates during order handling .....	36
Figure 14	All possible status transitions for actionStates .....	46
Figure 15	Handling multiple actions .....	47

## List of tables

Table 1	The acceptable values for the actionStatus field	45
Table 2	action blocking types	46

# 1 Foreword

The interface was established in cooperation between the Verband der Automobilindustrie e.V. (German abbreviation VDA) and Verband Deutscher Maschinen-und Anlagenbau e.V. (German abbreviation VDMA). The aim of both parties is to create an 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: <http://github.com/vda5050/vda5050>.

## 2 Objective of the document

The objective of the interface is to simplify the connection of new vehicles to an existing master control and thus to integrate into an existing automated guided vehicles (AGV) system when used in the automotive industry and to enable parallel operation with AGV from different manufacturers and conventional systems (inventory systems) in the same working environment.

Uniform interface between master control and AGV shall be defined. In detail, 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).

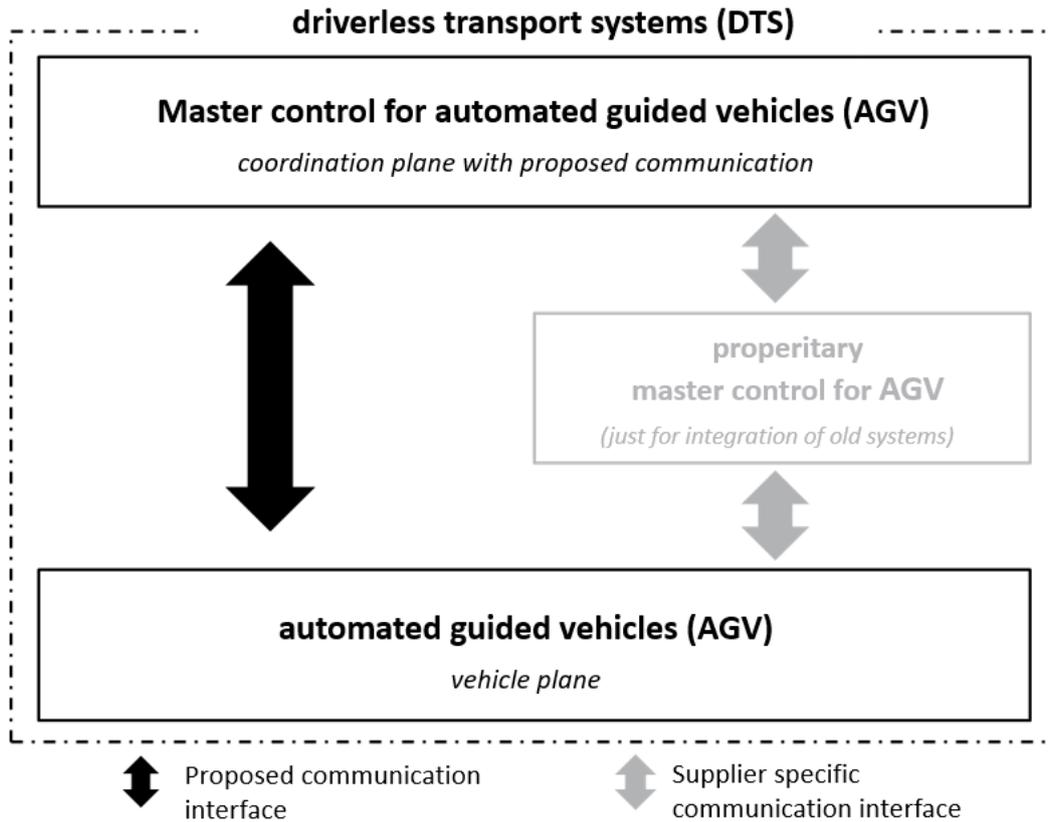


Figure 1 Integration of DTS inventory systems

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., for exchanging map information, 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 AGV with different characteristics (e.g., underrun tractor or forklift AGV) to communicate with master control in uniform language. This creates the basis for operating any combination of AGV in a master control. The master control provides orders and coordinates the AGV 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 free navigating vehicles and guided vehicles.

In contrast to automated vehicles, autonomous vehicles solve problems that occur based on 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 free navigating vehicles as well as guided vehicles. However, as soon as the path planning is carried out on the vehicle itself, this document describes free 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 free 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	Description
VDI Guideline 2510	Driverless transport systems (DTS)
VDI Guideline 4451 Sheet 7	Compatibility of driverless transport systems (DTS) - DTS master control
DIN EN ISO 3691-4	Industrial Trucks Safety Requirements and Verification- Part 4: Driverless trucks and their systems

## 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 log 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

As shown in the information flow to the operation of AGV, there are at least the following participants (see Figure 2):

- the operator 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.

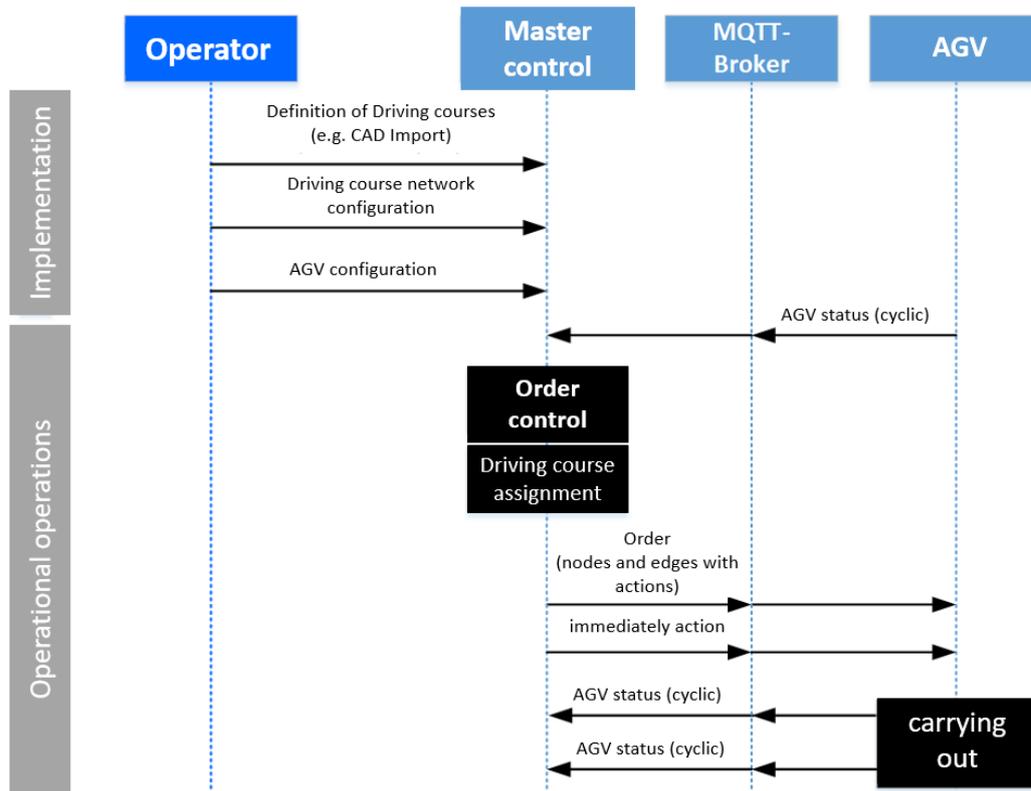


Figure 2 Structure of the Information Flow

During the implementation phase, the driverless transport systems (DTS) consisting of master control and AGV is set up. The necessary framework conditions are defined by the operator and the required information is either entered manually by him 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 taken over in 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 must communicate this information via the subtopic `factsheet` in a specific way that is defined in the AGV Factsheet section (chapter 6.15) of this document.

The configuration of routes and the route network described above is not part of this document. It forms 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 AGV
- Route calculation and guidance of the AGV (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 AGV are:

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

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

- Map configuration: The coordinate systems of the master control and the AGV must 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.

### 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
<b>bold</b>	Variable is a non-elementary data type (e.g., JSON-object or array) and defined separately
<i>italic</i>	Variable is optional
[Square brackets]	Variable (here arrayName) is an array of the data type included in the square brackets (here the data type is squareBrackets)

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

#### 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 AGV 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 must 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. If an AGV 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 must supply an AGV factsheet that is detailed in section 6.15 Topic "factsheet".

### 6.1.3 Notation of enumerations

Enumerations must be written in uppercase. This includes keywords such as the states of the actions (WAITING, FINISHED, etc...) or values of the "direction" field (LEFT, RIGHT, 443MHZ, etc...).

### 6.1.4 JSON Datatypes

Where possible, JSON data types must be used. A Boolean value is thus encoded by "true / false", NOT with an enumeration (TRUE, FALSE) or magic numbers.

## 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 in account by broker configuration.

To reduce the communication overhead, the MQTT QoS level 0 (Best Effort) is to be used for the topics `order`, `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, preceded by "v"
manufacturer	string	Manufacturer of the AGV (e.g., RobotCompany)
serialNumber	string	Unique AGV Serial Number consisting of the following characters: A-Z a-z 0-9 - . : -
topic	string	Topic (e.g. Order or System State) see Cap. 6.5

Note: Since the / character is used to define topic hierarchies, it must 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 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.ssZ (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 Subtopics for communication

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

Subtopic 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	mandatory	connection.schema

Subtopic name	Published by	Subscribed by	Used for	Implementation	Schema
			MQTT protocol level check of connection		
factsheet	AGV	master control	Setup of 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.

It is to be avoided that the master control has a separate graph representation for each type of AGV. Whenever possible, one location, e.g., a waiting position in front of fire door, should only have one node for all types of AGV. However, due to the different sizes and specifications of AGV, it might be necessary to deviate from this standard in certain situations.

**Representation in Master Control**

**Graph transmitted to AGV in topic order**

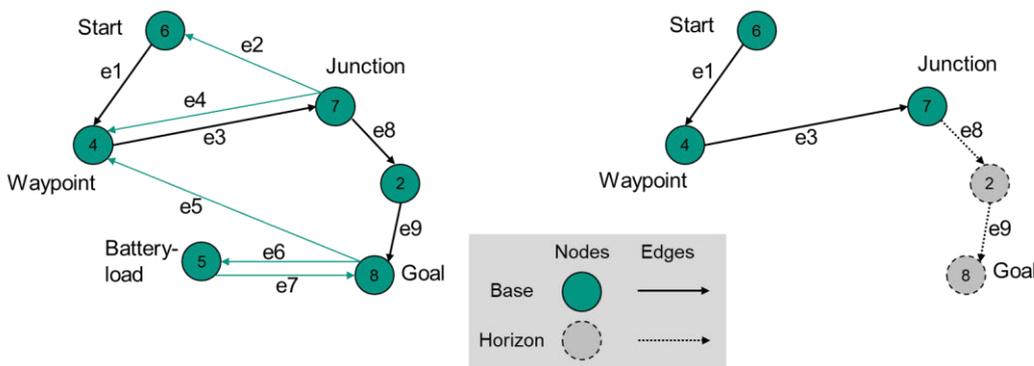


Figure 3 Graph representation in Master Control and graph transmitted in orders

The nodes and edges are passed as two lists in the order message. The lists order also governs in which sequence the nodes and edges must be traversed.

For a valid order, at least one node must be present. The number of acceptable edges is the number of nodes minus one, not more or less.

The first node of an order must 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 nodes 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 must not traverse it.

An edge only can be released, if both the start and 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

For traffic control the order-topic includes only the path to a decision point. Before reaching the decision point, the master control will send an updated path with additional path segments. To communicate to the AGV what it will most likely have to do after reaching the decision point, an order consists of two separate parts:

- Drive to the decision point "Base": The "Base" is the defined route that the AGV travels. All nodes and edges of the "Base" route have already been approved by the control panel for the vehicle.
- Estimated journey from the decision point "Horizon": The "Horizon" is the route that the AGV is likely to drive, if there is no traffic jam. The "Horizon" route has not yet been approved by the control panel. The AGV will initially only travel to the last junction of the "Base" route.

Since MQTT is an asynchronous protocol and transmission via wireless networks is not reliable, it is important to note, that the "base" cannot be changed. The master control can therefore assume that the "base" is executed by the AGV. A later section describes a procedure for cancelling an order, but this is also considered unreliable due to the communication restrictions mentioned above.

The master control has the possibility to change the driving commands of the "Horizon" route. Before the AGV arrives at the decision point via the "base" route, the master control will send an updated route to the AGV, which includes the other nodes. The procedure for changing the Horizon route is shown in Figure 4.

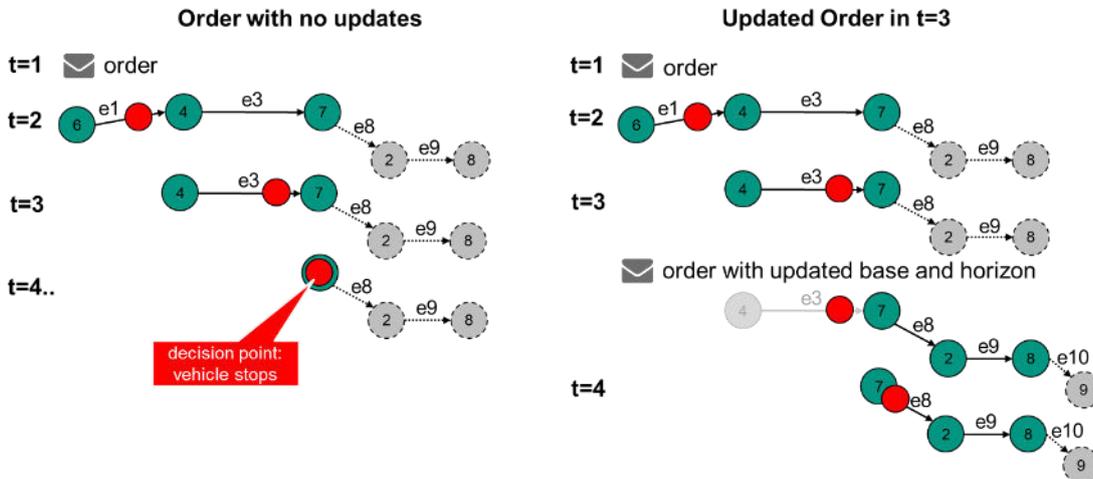


Figure 4 Procedure for changing the driving route "Horizon"

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: [
    6 {released: True},
    4 {released: True},
    7 {released: True},
    2 {released: False},
    8 {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 job 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 job update is reachable by executing the edges already known to the AGV.

```

{
  orderId: 1234,
  orderUpdateId: 1,
  nodes: [
    7 {released: True},
    2 {released: True},
    8 {released: True},
    9 {released: False}
  ],
  edges: [
    e8 {released: True},
    e9 {released: True},
    e10 {released: False}
  ]
}

```

*Figure 6 Pseudocode of an order update. Please look out for the change of the "orderUpdateId"*

This also aids in the event that an orderUpdate goes missing (because of 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 7 is the only base node that is sent again. Since the base cannot be changed, a retransmission of nodes 6 and 4 is not valid.

It is important, that the contents of the stitching node (node 7 in the example case) are not changed. For actions, deviation range, etc. the AGV must use the instructions provided in the first order (Figure 5, orderUpdateId 0).

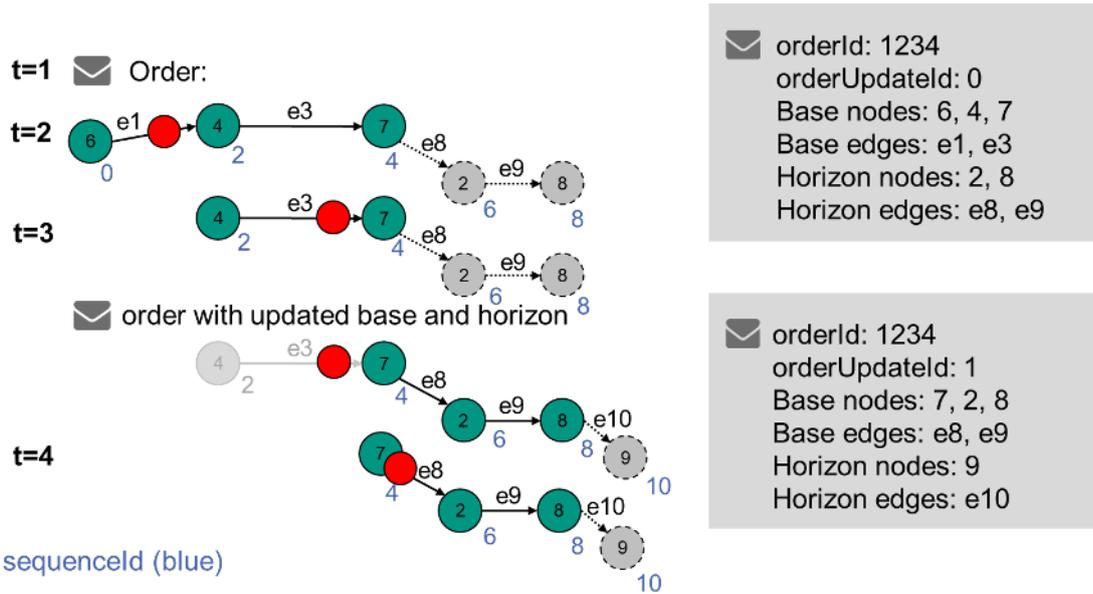


Figure 7 Regular update process - order extension

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 1 to 2 and then back to 1) a sequencId is assigned to the node and edge objects. This sequencId 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 sequencId 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 orderUpdate.

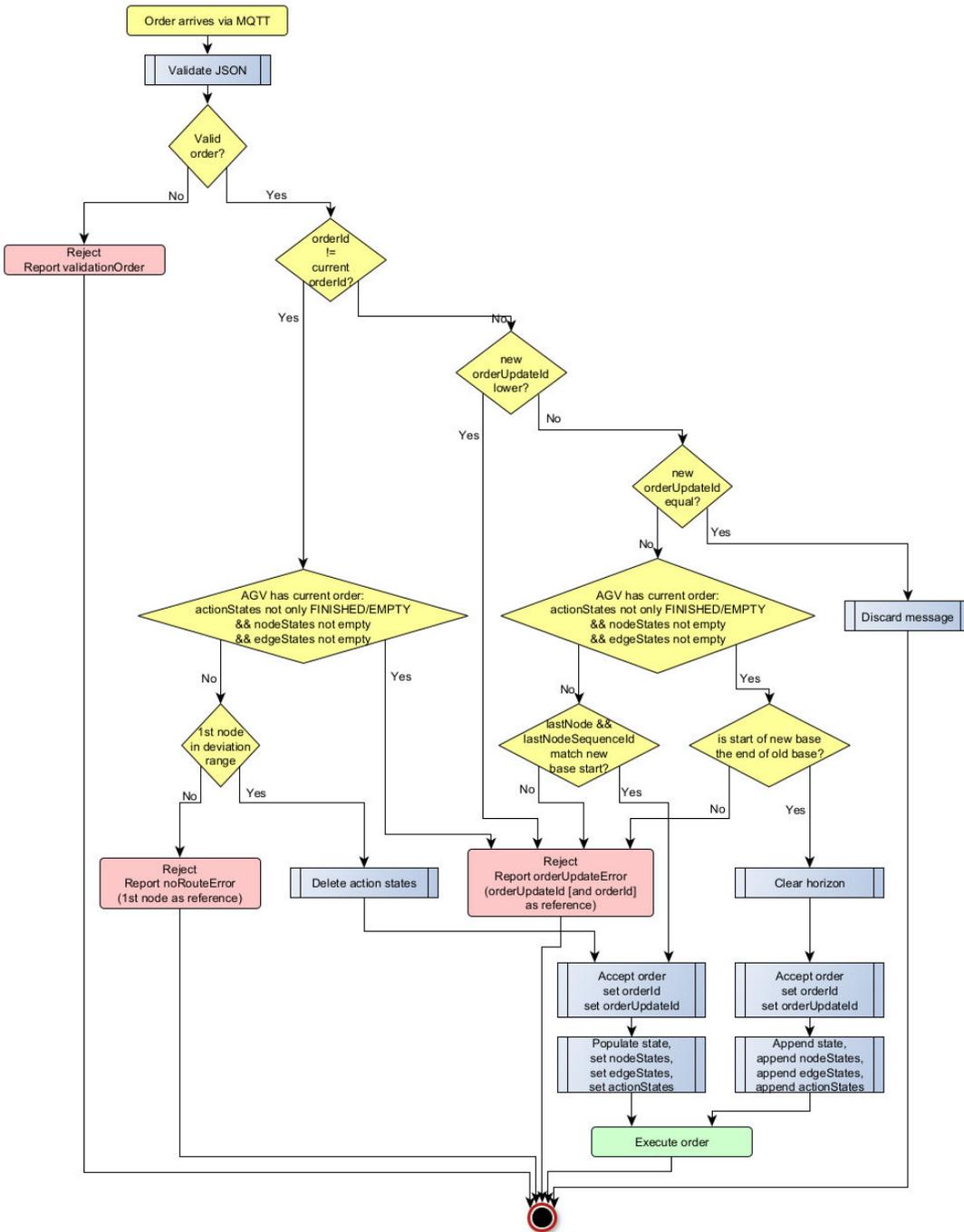


Figure 8 The process of accepting an order or orderUpdate

### 6.6.3 Order Cancellation (by Master Control)

In the event of an unplanned change in the base nodes, the order must 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 must be cancelled and should 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 must report “running”, until all actions are cancelled/finished. After all vehicle movements and all actions are stopped, the cancelOrder action status must report “finished”.

The orderId and orderUpdateId is kept.

Figure 9 shows the expected behavior for different AGV capabilities.

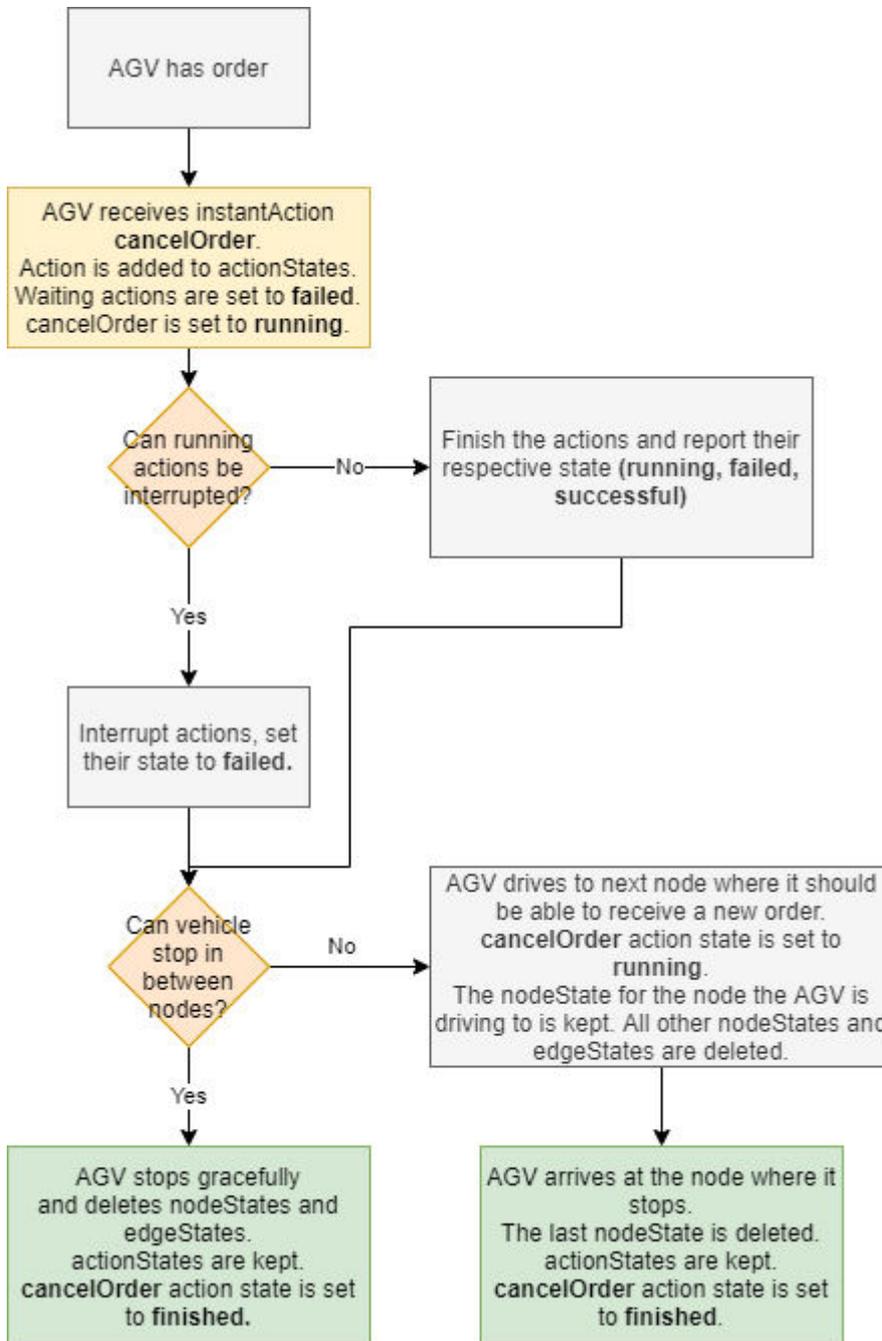


Figure 9 Expected behavior after a cancelOrder

### 6.6.3.1 Receiving a new order after cancellation

After the cancellation of an order, the vehicle must 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 8).

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 must 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 must 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 must realize that this node must 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 cancelled, the cancelOrder action must report as failed.

The AGV must report a "noOrderToCancel" error with the errorLevel set to warning. The actionId of the instantAction must be passed as an errorReference.

## 6.6.4 Order rejection

There are several scenarios, when an order must be rejected. These are explained in Figure 8.

### 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 must be reported until the vehicle has accepted a new order.

### 6.6.4.2 Vehicle receives an order with actions it cannot perform (e.g. lifting height higher than maximum lifting height, or lifting actions although no stroke is installed), or with fields that it cannot use (e.g. Trajectory)

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 must 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 the 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 sends the order again, because the status message came too late, and the master control could not verify that the first order was received.

## 6.6.5 Maps

To ensure consistent navigation among different types of AGV, the position is always specified in reference to the local map coordinate system (see Figure 10). For the differentiation between different levels 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 skywards. This is in accordance with chapter 2.11 in DIN ISO 8855.

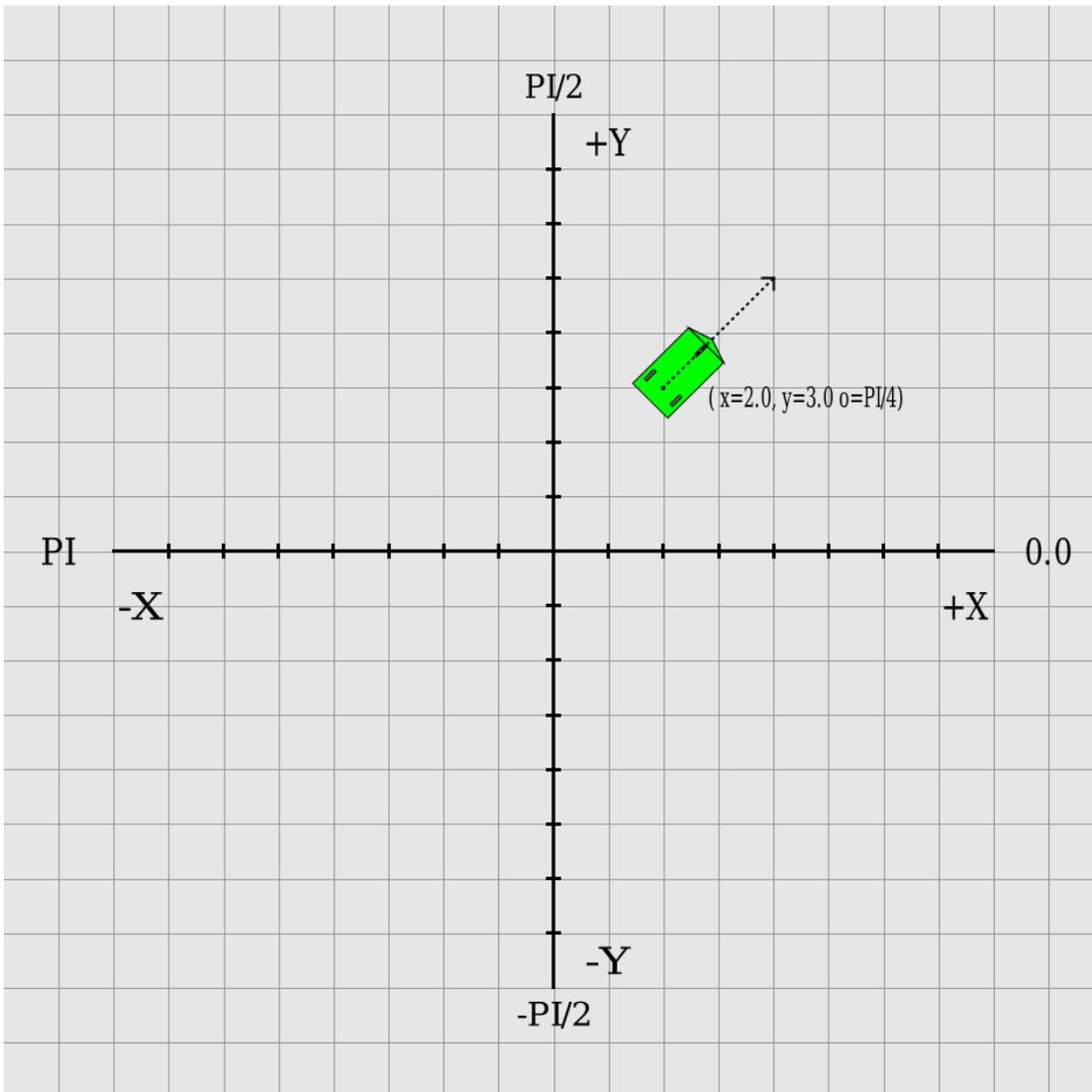


Figure 10 Coordinate system with sample AGV and orientation

The X, Y and Z coordinates must be in meters. The orientation must be in radians and must be within  $+\pi$  and  $-\pi$ .

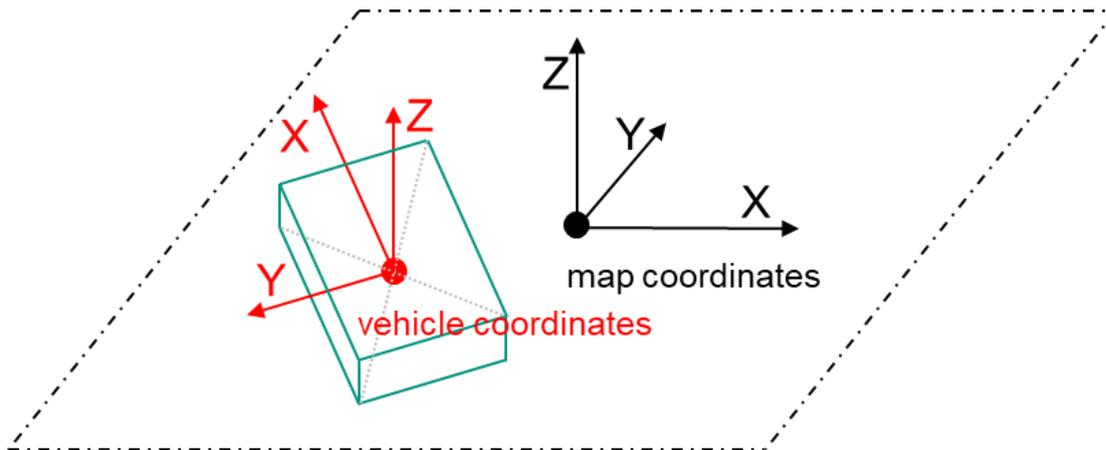


Figure 11 Coordinate systems for map and vehicle

## 6.7 Implementation of the order 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.ssZ (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 AGV do not understand zones. Do not add to message, if no zones are used.
nodes [node]		array	Array of nodes objects to be traversed for fulfilling the order. One node is enough for a valid order. Leave edge list empty for 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 list empty for that case.
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 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.
<i>nodeDescription</i>		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.
<i>nodePosition</i>		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.
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 must 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.

Object structure	Unit	Data type	Description
<i>theta</i>	rad	float64	<p>Range: [-Pi ... Pi]</p> <p>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.  If previous edge disallows rotation, the AGV must 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.</p>
<i>allowedDeviationXY</i>		float64	<p>Indicates how exact an AGV has to drive over a node in order for it to count as traversed.</p> <p>If = 0: no deviation is allowed (no deviation means within the normal tolerance of the AGV manufacturer).</p> <p>If &gt; 0: allowed deviation-radius in meters. If the AGV passes a node within the deviation-radius, the node is considered to have been traversed.</p>
<i>allowedDeviationTheta</i>		float64	<p>Range: [0 ... Pi]</p> <p>Indicates how big the deviation of theta angle can be.  The lowest acceptable angle is theta - allowedDeviationTheta and the highest acceptable angle is theta + allowedDeviationTheta.</p>
<i>mapId</i>		string	<p>Unique identification of the map in 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.</p>
<i>mapDescription</i> }		string	Additional information on the map.
<i>action</i> {		JSON-object	Describes an action that the AGV can perform.
<i>actionType</i>		string	Name of action as described in the first column of "Actions and Parameters". Identifies the function of the action.
<i>actionId</i>		string	Unique ID to identify the action and map them to the actionState in the state. Suggestion: Use UUIDs.
<i>actionDescription</i>		string	Additional information on the action

Object structure	Unit	Data type	Description
blockingType		string	Enum {NOTE, SOFT, HARD}; "NONE"- allows driving and other actions; "SOFT"- allows other actions, but not driving; "HARD"- is the only allowed action at that time.
<i>actionParameters</i> <i>[actionParameter]</i>  }		array	Array of actionParameter-objects for the indicated action, e.g., deviceId, loadId, external Triggers.  See "Actions and Parameters"
edge {		JSON-object	Directional connection between two nodes.
edgeId		string	Unique edge identification.
sequenceId		Integer	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.
<i>edgeDescription</i>		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 startNode.
endNodeId		string	nodeId of endNode.
<i>maxSpeed</i>	m/s	float64	Permitted maximum speed on the edge. Speed is defined by the fastest measurement of the vehicle.
<i>maxHeight</i>	m	float64	Permitted maximum height of the vehicle, including the load, on edge.
<i>minHeight</i>	m	float64	Permitted minimal height of the load handling device on the edge.

Object structure	Unit	Data type	Description
<i>orientation</i>	rad	float64	<p>Orientation of the AGV on the edge. The value <i>orientationType</i> 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 = forwards and <math>\pi</math> = backwards. Example: orientation <math>\pi/2</math> rad will lead to a rotation of 90 degrees.</p> <p>If AGV starts in 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.</p> <p>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.</p>
<i>orientationType</i>		string	<p>Enum {GLOBAL, TANGENTIAL}: "GLOBAL"- relative to the global project specific map coordinate system; "TANGENTIAL"- tangential to the edge.</p> <p>If not defined, the default value is "TANGENTIAL".</p>
<i>direction</i>		string	<p>Sets direction at junctions for line-guided or wire-guided vehicles, to be defined initially (vehicle-individual). Examples: left, right, straight, 433MHz.</p>
<i>rotationAllowed</i>		boolean	<p>"true": rotation is allowed on the edge. "false": rotation is not allowed on the edge.</p> <p>Optional: No limit, if not set.</p>
<i>maxRotationSpeed</i>	rad/s	float64	<p>Maximum rotation speed</p> <p>Optional: No limit, if not set.</p>
<i>trajectory</i>		JSON-object	<p>Trajectory JSON-object for this edge as a NURBS. Defines the curve, on which the AGV should move between startNode and endNode.</p> <p>Optional: Can be omitted, if AGV cannot process trajectories or if AGV plans its own trajectory.</p>

Object structure	Unit	Data type	Description
<i>length</i>	m	float64	Length of the path from startNode to endNode  Optional: This value is used by line-guided AGVs to decrease their speed before reaching a stop position.
actions [action]  }		array	Array of actionIds 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 will stop and the state before entering the edge will be restored.
trajectory {		JSON-object	
<i>degree</i>		float64	Range: [1 ... infinity]  Defines the number of control points that influence any given point on the curve. Increasing the degree increases continuity.  If not defined, the default value is 1.
knotVector [float64]		array	Range: [ 0.0 ... 1.0]  Sequence of parameter values that determines where and how the control points affect the NURBS curve.  knotVector has size of number of control points + degree + 1.
controlPoints [controlPoint]  }		array	List of JSON controlPoint objects defining the control points of the NURBS, which includes the beginning and end point.
controlPoint {		JSON-object	
x		float64	X coordinate described in the world coordinate system.
y		float64	Y coordinate described in the world coordinate system.
<i>weight</i>		float64	Range: (0 ... infinity)  The weight, with which this control point pulls on the curve. When not defined, the default will be 1.0.
}			

## 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 6.9).

Actions that are to be executed on an edge must only run while the AGV is on the edge (see 6.10.2).

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 must be used by the AGV, if the AGVs capabilities map to the action description. If there is a sensible way to use the defined parameters, they must 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 must be used by master control.

## 6.8.1 Predefined action definition, their parameters, effects, and scope

general					scope			
action	counter action	Description	important	Parameter	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
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	yes	yes (Elevator)	no
stateRequest	-	Requests the AGV to send a new state report.	yes	-	-	yes	no	no

general					scope			
action	counter action	Description	important	Parameter	linked state	instant	node	edge
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). 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.	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 forklifts side(string) (optional) e.g. conveyor	.load	no	yes	yes
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.	yes	stationType(string, optional) stationName(string, optional)	-	no	yes	yes

general					scope			
action	counter action	Description	important	Parameter	linked state	instant	node	edge
		On an edge, AGV will e.g. align on stationary equipment while traversing an edge. InstantAction: AGV starts positioning exactly on a target.						
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	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 can not 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 can not be deactivated for some reason (e.g., overwritten by hardware switch).
startCharging	-	Activation of the charging process is in progress	-	The charging process is started.	The charging process could not be started for

action	initializing	running	paused	finished	failed
		(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 can not be reset. General localization problems should correspond with an error.
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.

<b>action</b>	<b>initializing</b>	<b>running</b>	<b>paused</b>	<b>finished</b>	<b>failed</b>
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 must 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 chapter 7 Best practices.

Object structure	Data type	Description
headerId	uint32	header ID of the message. The headerId is defined per topic and incremented by 1 with each sent (but not necessarily received) message.
timestamp	string	Timestamp (ISO 8601, UTC); YYYY-MM-DDTHH:mm:ss.ssZ (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 (see chapter 6.7).

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 12 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 or actionStates

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 AVG plans the path by itself, it must communicate its calculated trajectory (including base and horizon) in the form of a 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 must 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 must traverse.

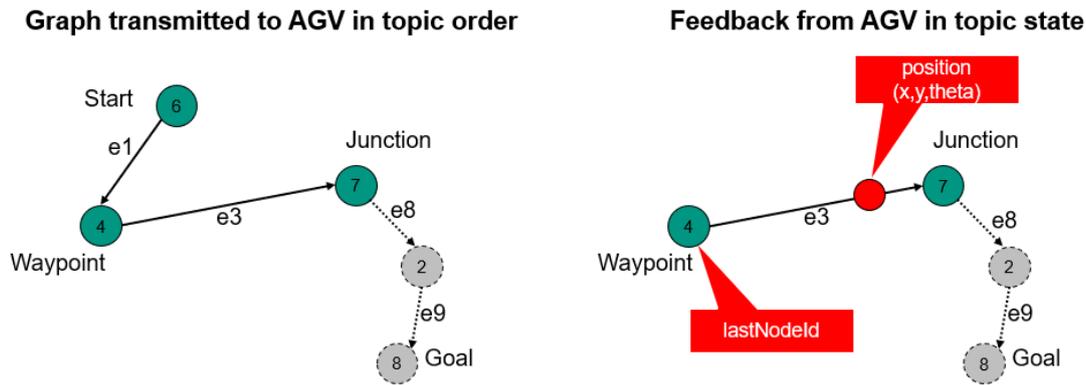


Figure 12 Order Information provided by the state topic.  
Only the ID of the last node and the remaining nodes and edges are transmitted

## 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 `deviationRangeXY` and its orientation within `deviationRangeTheta`.

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

As soon as the AGV reports the node as traversed, the AGV must 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 must then be removed from the `edgeStates` and the actions that were active on the edge must be finished.

The traversal of the node also marks the moment, when the AGV enters the following edge, if there is one. The edges actions must 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.

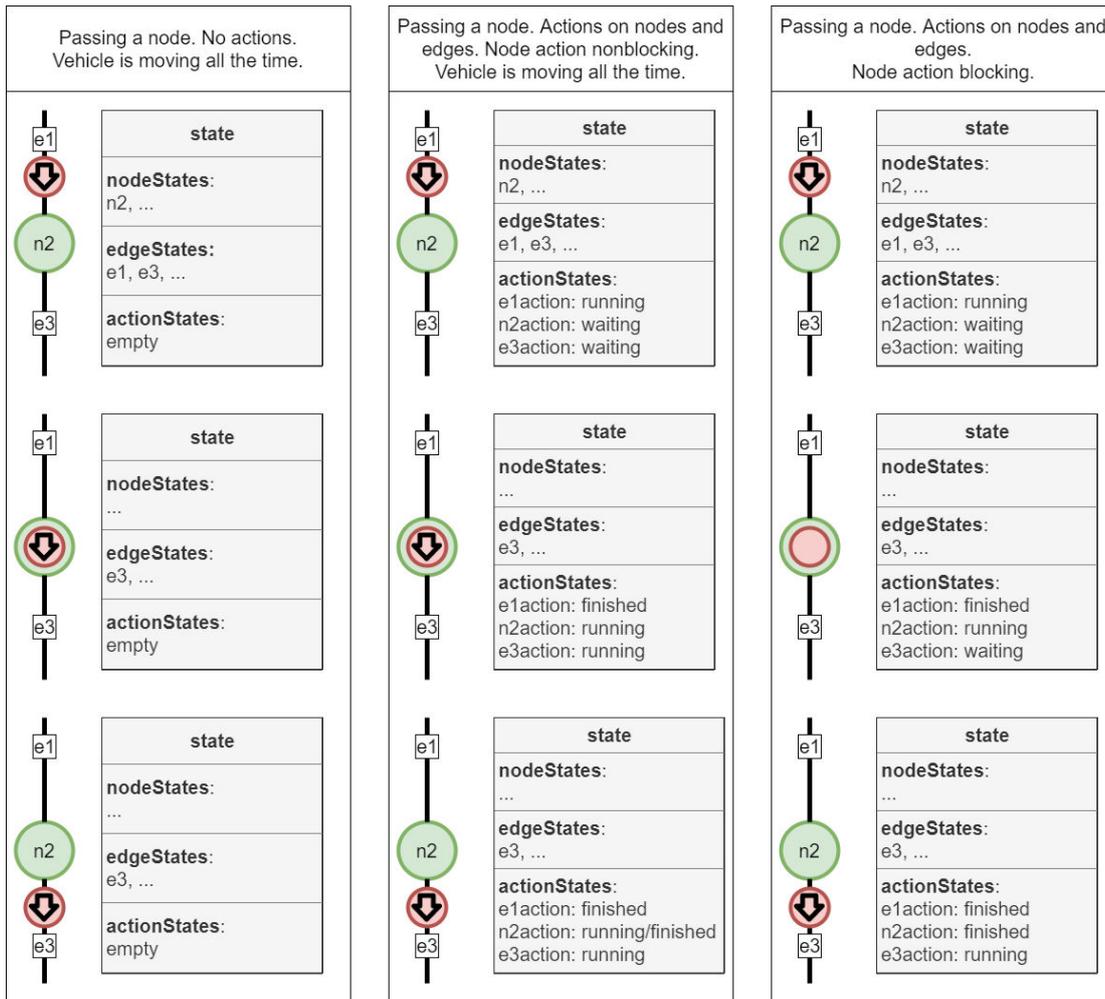


Figure 13 nodeStates, edgeStates, actionStates during order handling

### 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 must not use the info messages for logic, it must 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

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.ssZ (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	Unique order identification of the current order or the previous 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.
<i>zoneSetId</i>		string	Unique ID of the zone set, that the AGV currently uses for path planning. Must be the same as the one used in the order, otherwise the AGV has to reject 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 list if idle)
edgeStates [edgeState]		array	Array of edgeState-Objects, that need to be traversed for fulfilling the order (empty list if idle)

Object structure	Unit	Data type	Description
<i>agvPosition</i>		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.
<i>velocity</i>		JSON-object	The AGV velocity in vehicle coordinates.
<i>loads [load]</i>		array	Loads, that are currently handled by the AGV.  Optional: If AGV cannot determine load state, leave the array out of the state. If the AGV can determine the load state, but the array is empty, the AGV is considered unloaded.
<i>driving</i>		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.
<i>paused</i>		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.
<i>newBaseRequest</i>		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.
<i>distanceSinceLastNode</i>	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	<p>Contains a list of the current actions and the actions, which are yet to be finished.</p> <p>This may include actions from previous nodes, that are still in progress.</p> <p>When an action is completed, an updated state message is published with actionStatus set to finished and if applicable with the corresponding resultDescription.</p> <p>The action state is kept until a new order is received.</p>
batteryState		JSON-object	Contains all battery-related information.
operatingMode		string	<p>Enum {AUTOMATIC, SEMIAUTOMATIC, MANUAL, SERVICE, TEACHIN}</p> <p>For additional information, see the table OperatingModes in the chapter 6.10.6.</p>
errors [error]		array	<p>Array of error-objects.</p> <p>All active errors of the AGV should be in the list.</p> <p>An empty array indicates that the AGV has no active errors.</p>
<i>information [info]</i>		array	<p>Array of info-objects.</p> <p>An empty array indicates, that the AGV has no information.</p> <p>This should only be used for visualization or debugging – it must not be used for logic in master control.</p>
safetyState		JSON-object	Contains all safety-related information.
nodeState {		JSON-object	
nodeId		string	Unique node identification.
sequenceId		uint32	sequenceId to discern multiple nodes with same nodeId.
<i>nodeDescription</i>		string	Additional information on the node.
<i>nodePosition</i>		JSON-object	<p>Node position.</p> <p>The object is defined in chapter 6.6</p> <p>Optional:</p> <p>Master control has this information. Can be sent additionally, e. g. for debugging purposes.</p>

Object structure	Unit	Data type	Description
released }		boolean	“true” indicates that the node is part of the base. “false” indicates that the node is part of the horizon.
edgeState {		JSON-object	
edgeId		string	Unique edge identification.
sequenceId		uint32	sequenceId to differentiate between multiple edges with the same edgeId.
<i>edgeDescription</i>		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.
<i>trajectory</i> }		JSON-object	The trajectory is to be communicated as a NURBS and is defined in chapter 6.4  Trajectory segments are defined from the point, where the AGV starts to enter the edge, to the point, where it reports, that the next node was traversed.
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.
<i>localizationScore</i>		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.
<i>deviationRange</i>	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.

Object structure	Unit	Data type	Description
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, 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.
<i>mapDescription</i> }		string	Additional information on the map.
velocity {		JSON-object	
vx	m/s	float64	The AVGs velocity in its x direction.
vy	m/s	float64	The AVGs velocity in its y direction.
<i>omega</i> }	Rad/s	float64	The AVGs turning speed around its z axis.
load {		JSON-object	
<i>loadId</i>		string	Unique identification number 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.
<i>loadType</i>		string	Type of load.
<i>loadPosition</i>		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

Object structure	Unit	Data type	Description
<i>boundingBoxReference</i>		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.
<i>loadDimensions</i>		JSON-object	Dimensions of the load's bounding box in meters.
<i>weight</i> }	kg	float64	Range: [0.0 ... infinity)  Absolute weight of the load measured in kg.
<i>boundingBoxReference</i> {		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.
<i>x</i>		float64	x-coordinate of the point of reference.
<i>y</i>		float64	y-coordinate of the point of reference.
<i>z</i>		float 64	z-coordinate of the point of reference.
<i>theta</i> }		float64	Orientation of the loads bounding box. Important for tugger, trains, etc.
<i>loadDimensions</i> {		JSON-object	Dimensions of the load's bounding box in meters.
<i>length</i>	m	float64	Absolute length of the load's bounding box.
<i>width</i>	m	float64	Absolute width of the load's bounding box.
<i>height</i>  }	m	float64	Absolute height of the load's bounding box.  Optional:  Set value only if known.
<i>actionState</i> {		JSON-object	
<i>actionId</i>		string	action_ID
<i>actionType</i>		string	actionType of the action.  Optional: Only for informational or visualization purposes. Order knows the type.
<i>actionDescription</i>		string	Additional information on the current action.

Object structure	Unit	Data type	Description
actionStatus		string	Enum {WAITING; INITIALIZING; RUNNING; PAUSED; FINISHED; FAILED}  WAITING: waiting for the trigger (passing the mode, entering the edge) PAUSED: paused by instantAction or external trigger FAILED: action could not be performed.
resultDescription  }		string	Description of the result, e.g., the result of a RFID-read.  Errors will be transmitted in errors.  Examples for results are given in 6.5
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).
batteryVoltage	V	float64	Battery Voltage.
batteryHealth	%	int8	Range: [0 .. 100]  State of Health.
charging		boolean	“true”: charging in progress. “false”: AGV is currently not charging.
reach  }	m	uint32	Range: [0 ... infinity)  Estimated reach with current State of Charge.
error {		JSON-object	
errorType		string	Type/name of error
errorReferences [errorReference]		array	Array of references to identify the source of the error (e.g., headerId, orderId, actionId, etc.). For additional information see „Best practices“ chapter 7.
errorDescription		string	Error description.
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).
errorReference {		JSON-object	

Object structure	Unit	Data type	Description
referenceKey		string	References the type of reference (e.g., headerId, orderId, actionId, etc.).
referenceValue }		string	References the value, which belongs to the reference key.
info {		JSON-object	
infoType		string	Type/name of information.
<i>infoReferences</i> [ <i>infoReference</i> ]		array	Array of references.
<i>infoDescription</i>		string	Info of description.
infoLevel  }		string	Enum {DEBUG,INFO}  DEBUG: used for debugging. INFO: used for visualization.
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.
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 has 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 "states".

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 send 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 personal 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.

## 6.11 actionStates

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

`actionStates` describe in the field `actionStatus` at which stage of the actions lifecycle the action is.

Table 1 describes, which value the enum `actionStatus` can hold.

<b>actionStatus</b>	<b>Description</b>
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 <code>instantAction</code> or external trigger (pause button on AGV)
FINISHED	The action is finished. A result is reported via the <code>resultDescription</code>
FAILED	Action could not be finished for whatever reason.

*Table 1 The acceptable values for the `actionStatus` field*

A state transition diagram is provided in Figure 14.

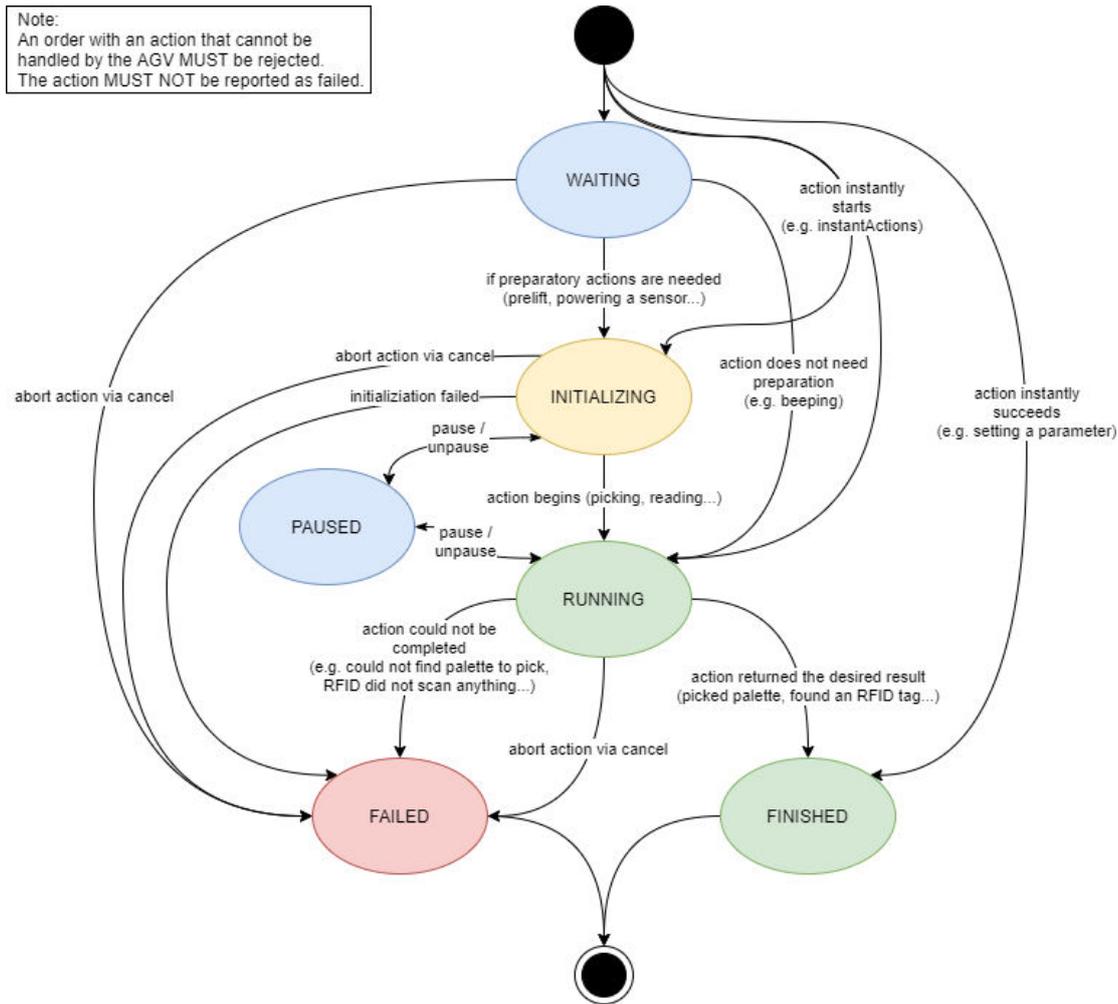


Figure 14 All possible status transitions for actionStates

## 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 2.

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 must not drive.
HARD	Action must not be executed in parallel with other actions. Vehicle must not drive.

Table 2 action blocking types

If there are multiple actions on the same node with different blocking types, Figure 15 describes how the AGV should handle these actions.

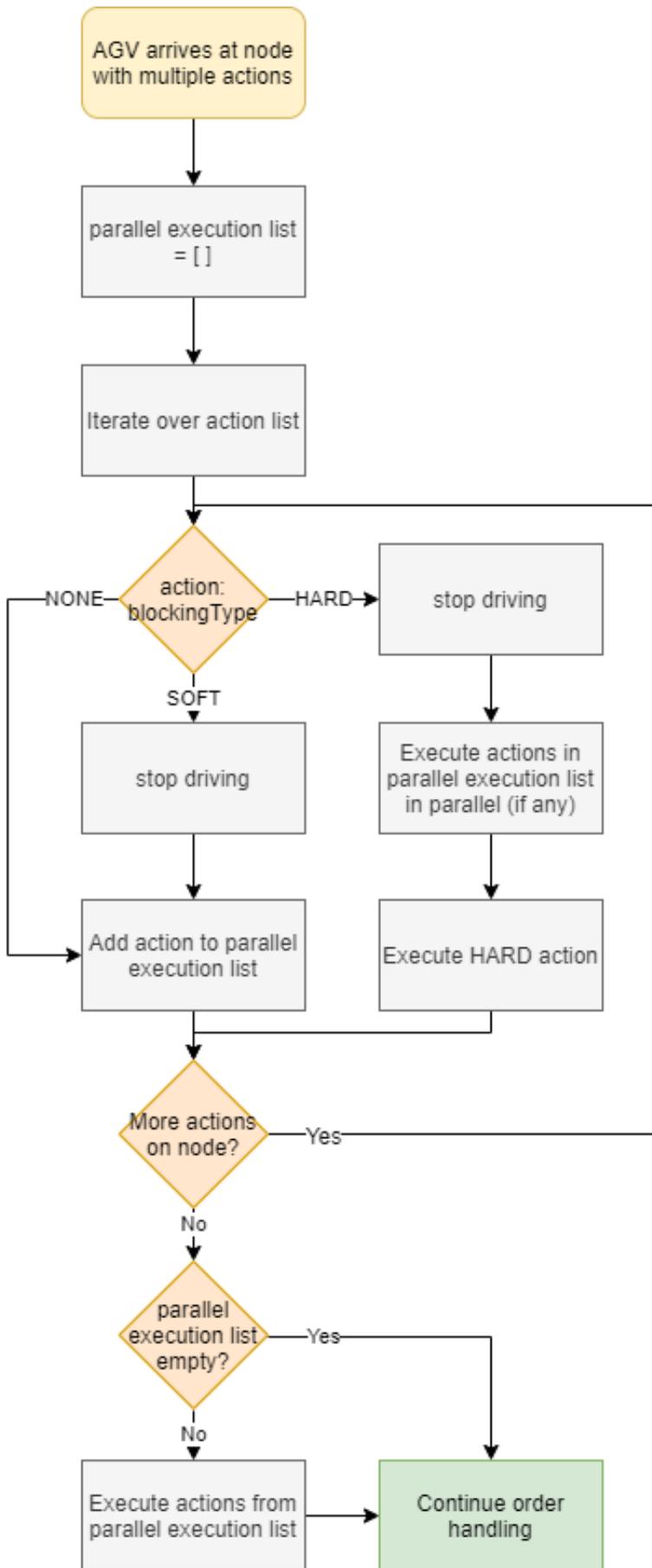


Figure 15 Handling multiple actions

## 6.13 Topic "visualization"

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

The structure of the position message is the same as the position and velocity message in the state. For additional information see chapter 6.7 Implementation. 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.ssZ(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 { <code>ONLINE</code> , <code>OFFLINE</code> , <code>CONNECTIONBROKEN</code> }  <code>ONLINE</code> : connection between AGV and broker is active.  <code>OFFLINE</code> : connection between AGV and broker has gone offline in a coordinated way.  <code>CONNECTIONBROKEN</code> : 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 a 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 both intended as 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.ssZ(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.

Field	data type	description
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.
loadSpecification	JSON-object	Abstract specification of load capabilities.
localizationParameters	JSON-object	Detailed specification of localization.

### 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.
<i>seriesDescription</i>	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	List 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.
accelerationMax	float64	[m/s <sup>2</sup> ] Maximum acceleration with maximum load.
decelerationMax	float64	[m/s <sup>2</sup> ] 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.
<i>msgLen</i>	uint32	Maximum MQTT message length
<i>topicSerialLen</i>	uint32	Maximum length of serial-number part in MQTT-topics.  Affected parameters: order.serialNumber instantActions.serialNumber state.SerialNumber visualization.serialNumber connection.serialNumber
<i>topicElemLen</i>	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
<i>idLen</i>	uint32	Maximum length of ID-Strings.  Affected parameters: order.orderId order.zoneSetId node.nodeId nodePosition.mapId action.actionId edge.edgId edge.startNodeId edge.endNodeId
<i>idNumericalOnly</i>	boolean	If "true" ID-strings need to contain numerical values only.
<i>enumLen</i>	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
<i>loadIdLen</i>	uint32	Maximum length of loadId Strings
}		
maxArrayLens {	JSON-object	Maximum lengths of arrays.
<i>order.nodes</i>	uint32	Maximum number of nodes per order processable by the AGV.
<i>order.edges</i>	uint32	Maximum number of edges per order processable by the AGV.
<i>node.actions</i>	uint32	Maximum number of actions per node processable by the AGV.
<i>edge.actions</i>	uint32	Maximum number of actions per edge processable by the AGV.
<i>actions.actionsParameters</i>	uint32	Maximum number of parameters per action processable by the AGV.
<i>instantActions</i>	uint32	Maximum number of instant actions per message processable by the AGV.
<i>trajectory.knotVector</i>	uint32	Maximum number of knots per trajectory processable by the AGV.
<i>trajectory.controlPoints</i>	uint32	Maximum number of control points per trajectory processable by the AGV.

Field	data type	description
<i>state.nodeStates</i>	uint32	Maximum number of nodeStates sent by the AGV, maximum number of nodes in base of AGV.
<i>state.edgeStates</i>	uint32	Maximum number of edgeStates sent by the AGV, maximum number of edges in base of AGV.
<i>state.loads</i>	uint32	Maximum number of load-objects sent by the AGV.
<i>state.actionStates</i>	uint32	Maximum number of actionStates sent by the AGV.
<i>state.errors</i>	uint32	Maximum number of errors sent by the AGV in one state-message.
<i>state.information</i>	uint32	Maximum number of informations sent by the AGV in one state-message.
<i>error.errorReferences</i>	uint32	Maximum number of error references sent by the AGV for each error.
<i>information.infoReferences</i>	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.
<i>defaultStateInterval</i>	float32	[s], Default interval for sending state-messages, <i>if not defined, the default value from the main document is used.</i>
<i>visualizationInterval</i>	float32	[s], Default interval for sending messages on visualization topic.
}		

### agvProtocolFeatures

This JSON object defines actions and parameters which are supported by the AGV.

Field	data type	description
optionalParameters [optionalParameter]	Array of JSON-object	List 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. " <i>order.nodes.nodePosition.allowedDeviationTheta</i> ".

Field	data type	description
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.
<i>description</i>	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' must contain unsigned interger-numbers only.  NURBS-Support is limited to straight lines and circle segments.
}		
agvActions [agvAction]	Array of JSON-object	List of all actions with parameters supported by this AGV. This includes standard actions specified in VDA5050 and manufacturer-specific actions.
{		
actionType	string	Unique actionType corresponding to action.actionType.
<i>actionDescription</i>	string	Free-form text: description of the action.
actionScopes	array of enum	List of allowed scopes for using this action-type.  INSTANT: usable as instantAction. NODE: usable on nodes. EDGE: usable on edges.  For example: [ „INSTANT“, „NODE“ ]
<i>actionParameters</i> [actionParameter]	Array of JSON-object	List of parameters If not defined, the action has no parameters
{		
key	string	Key-String for Parameter.
valueDataType	enum	Data type of Value, possible data types are: BOOL, NUMBER, INTEGER, FLOAT, STRING, OBJECT, ARRAY.
<i>description</i>	string	Free-form text: description of the parameter.
<i>isOptional</i>	boolean	"true": optional parameter.
}		
<i>resultDescription</i>	string	Free-form text: description of the resultDescription.
}		

## agvGeometry

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

Field	data type	description
<i>wheelDefinitions</i> [ <i>wheelDefinition</i> ]	Array of JSON-object	List of wheels, containing wheel-arrangement and geometry.
{		
type	enum	Wheel type DRIVE, CASTER, FIXED, MECANUM.
isActiveDriven	boolean	"true": wheel is actively driven (de: angetrieben).
isActiveSteered	boolean	"true": wheel is actively steered (de: aktiv gelenkt).
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.
}		
<i>envelopes2d</i> [ <i>envelope2d</i> ]	Array of JSON-object	List of AGV-envelope curves in 2D (german: „Hüllkurven“), 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 [ <i>polygonPoint</i> ]	Array of JSON-object	Envelope curve as a x/y-polygon polygon is assumed as closed and must 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
<i>envelopes3d [envelope3d]</i>	Array of JSON-object	List of AGV-envelope curves in 3D (german: „Hüllkurven“).
{		
<i>set</i>	string	Name of the envelope curve set.
<i>format</i>	string	Format of data, e.g., DXF.
<i>data</i>	JSON-object	3D-envelope curve data, format specified in 'format'.
<i>url</i>	string	Protocol and url-definition for downloading the 3D-envelope curve data, e.g. ftp://xxx.yyy.com/ac4dgvhoif5tghji.
<i>description</i>	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
<i>loadPositions</i>	Array of String	List of load positions / load handling devices. This lists contains the valid values for the parameter “state.loads[].loadPosition” and for the action parameter “lhd” of the actions pick and drop. <i>If this list doesn't exist or is empty, the AGV has no load handling device.</i>
<i>loadSets [loadSet]</i>	Array of JSON-object	list of load-sets that can be handled by the AGV
{		
<i>setName</i>	string	Unique name of the load set, e.g., DEFAULT, SET1, etc.
<i>loadType</i>	string	Type of load, e.g., EPAL, XLT1200, etc.
<i>loadPositions</i>	Array of String	List of load positions btw. load handling devices, this load-set is valid for. <i>If this parameter does not exist or is empty, this load-set is valid for all load handling devices on this AGV.</i>
<i>boundingBoxReference</i>	JSON-object	Bounding box reference as defined in parameter loads[] in state-message.
<i>loadDimensions</i>	JSON-object	Load dimensions as defined in parameter loads[] in state-message.
<i>maxWeight</i>	float64	[kg], maximum weight of loadtype.
<i>minLoadhandlingHeight</i>	float64	[m], minimum allowed height for handling of this load-type and –weight references to boundingBoxReference.

<b>Field</b>	<b>data type</b>	<b>description</b>
<i>maxLoadhandlingHeight</i>	float64	[m], maximum allowed height for handling of this load-type and –weight references to boundingBoxReference.
<i>minLoadhandlingDepth</i>	float64	[m], minimum allowed depth for this load-type and –weight references to boundingBoxReference.
<i>maxLoadhandlingDepth</i>	float64	[m], maximum allowed depth for this load-type and –weight references to boundingBoxReference.
<i>minLoadhandlingTilt</i>	float64	[rad], minimum allowed tilt for this load-type and –weight.
<i>maxLoadhandlingTilt</i>	float64	[rad], maximum allowed tilt for this load-type and –weight.
<i>agvSpeedLimit</i>	float64	[m/s], maximum allowed speed for this load-type and –weight.
<i>agvAccelerationLimit</i>	float64	[m/s <sup>2</sup> ], maximum allowed acceleration for this load-type and –weight.
<i>agvDecelerationLimit</i>	float64	[m/s <sup>2</sup> ], maximum allowed deceleration for this load-type and –weight.
<i>pickTime</i>	float64	[s], approx. time for picking up the load
<i>dropTime</i>	float64	[s], approx. time for dropping the load.
<i>description</i>	string	Free-form text: description of the load handling set.
}		

## 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 fields `errorReference` (see [6.10.6](#)). 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, actions are designed as an array of JSON-Objects with key-value-pairs. Sample for the `actionParameter` of an action "someAction" with key-value-pairs for `stationType` and `loadType`:

```
"actionParameters": [
  {"key": "stationType", "value": "floor"},
  {"key": "loadType", "value": "pallet_eu"}
]
```

The reason for using the proposed scheme of "key": "actualKey", "value": "actualValue" is to keep the implementation generic. This was thoroughly and controversially discussed in multiple meetings.

## 8 Glossary

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

If you notice any errors, omissions or ambiguities in these recommendations, please contact VDA without delay so that these errors can be rectified.

Publisher German Association of the Automotive Industry  
Behrenstraße 35, 10117 Berlin  
[www.vda.de/en](http://www.vda.de/en)

Copyright German Association of the Automotive Industry

Reprint, also in extracts, is only permitted,  
if the source is stated.