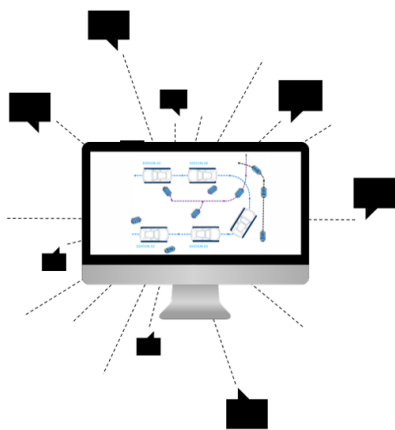


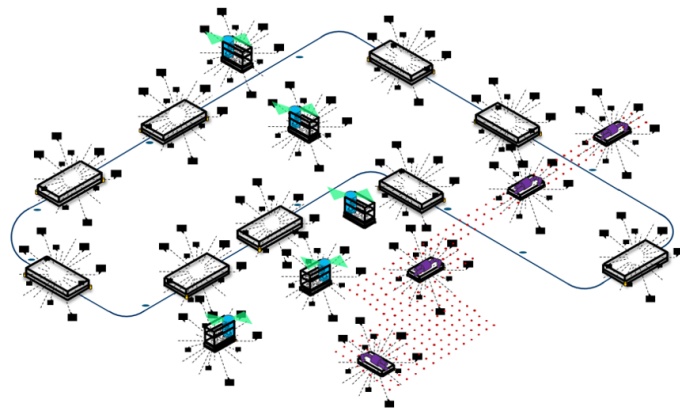
Interface for the communication between automated guided vehicles (AGV) and a master control

VDA 5050

Version 1.1, June 2020



control system



automated guided vehicles

Brief information

Definition of a communication interface for driverless transport systems (DTS).

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

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

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

Publisher Verband der Automobilindustrie e.v. (VDA)
Behrenstrasse 35, 10117 Berlin
www.vda.de

Copyright Association of the Automotive Industry (VDA)

Reproduction and any other form of reproduction is only permitted with specification of the source.

Belaystation June 2020

Version Version 1.1

List of content

1	Foreword.....	4
2	Objective of the document.....	5
3	Scope	6
3.1	Other applicable documents.....	7
4	Requirements and protocol definition.....	7
5	Process and content of communication.....	8
6	Protocol specification.....	9
6.1	Symbols of tables and meaning of formatting.....	9
6.1.1	Optional fields	10
6.1.2	Permitted characters and field lengths.....	10
6.1.3	Notation of enumerations.....	10
6.1.4	JSON Datatypes.....	10
6.2	MQTT connection handling, security and QoS.....	11
6.3	MQTT-Topic Levels.....	11
6.4	Protocol Header.....	11
6.5	Subtopics for communication.....	12
6.6	Topic: Order (from master control to AGV)	12
6.6.1	Concept and Logic	13
6.6.2	Orders and order updates	13
6.6.3	Order Cancellation (by Master Control)	18
6.6.4	Order rejection	20
6.6.5	Maps	20
6.7	Implementation of the order message	21
6.8	Actions.....	24
6.8.1	Predefined action definitions, their parameters, effects and scope	24
6.8.2	Predefined action definitions, description of their states.....	27
6.9	Topic: instantActions (from master control to AGV).....	28
6.10	Topic: State (from AGV to master control)	29
6.10.1	Concept and Logic	29
6.10.2	Traversal of nodes and entering/leaving edges, triggering of actions	30
6.10.3	Base request	31
6.10.4	Information	31
6.10.5	Errors.....	32
6.10.6	Implementation	32

6.11	actionStates.....	37
6.12	Action Blocking Types and sequence.	38
6.13	Topic “Visualization”.....	39
6.14	Topic „connection“.....	39
7	Best practice.....	40
7.1	Error reference.....	41
7.2	Format of parameters.....	41
8	Glossary.....	41
8.1	Definition.....	41

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) developed. The aim of both parties is to create a universally applicable interface. Proposals for changes to the interface shall be submitted to the VDA, are evaluated jointly with the VDMA and adopted into a new version status in the event of a positive decision.

2 Objective of the document

The objective of the document is to simplify the connection of new vehicles to an existing master control and thus to integrate into an existing AGV 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 the master control and the automated guided vehicles (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. Hardware and software are needed to increase autonomy
- Reduction of implementation time due to high "Plug & Play" capability, as required information (z. B. Cards) 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 manufacturer's independence using common interfaces between vehicle control and coordination level
- Integration of proprietary DTS inventory systems by implementing a vertical communication between the proprietary master control and the superordinate master control (cf. Figure 1)

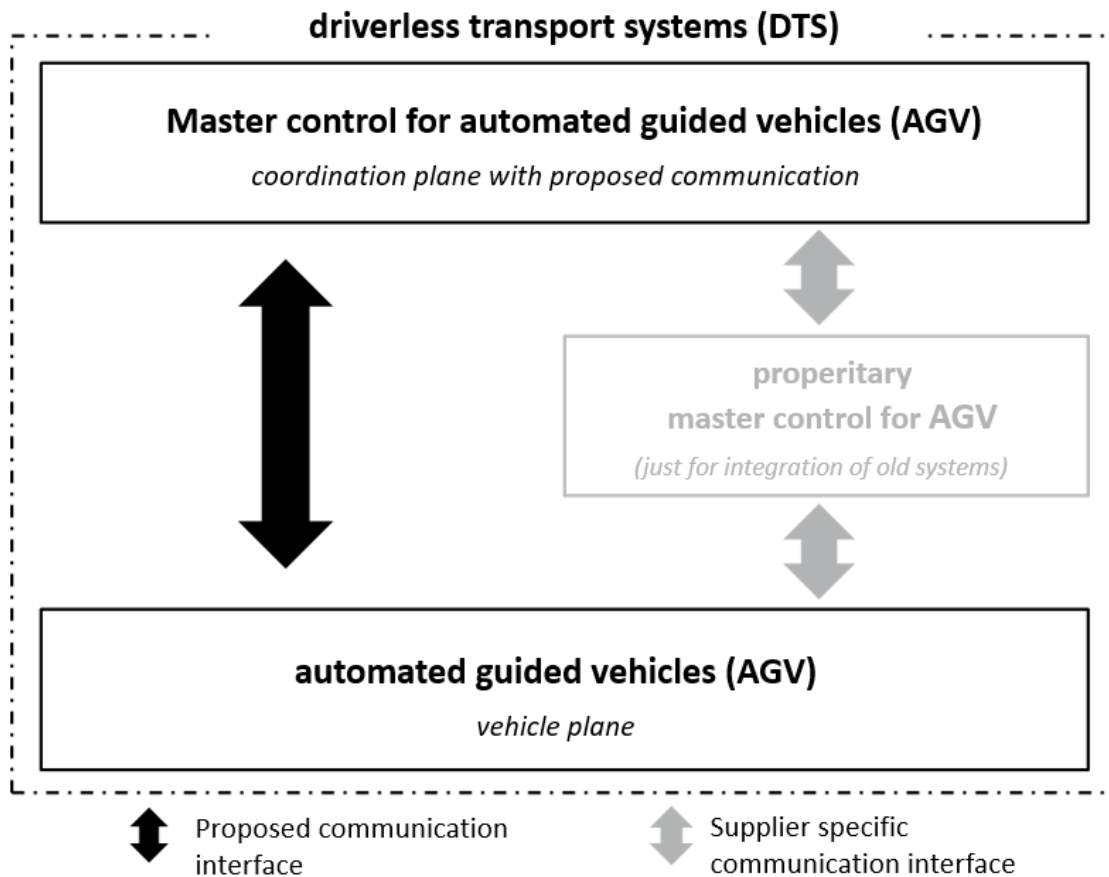


Figure 1 Integration of DTS inventory systems

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

Other interfaces required for operation between the AGV and the 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 document contains definitions and best practice regarding communication between automated guided vehicles (AGVs) and a master control. The aim is for AGVs to speak with different characteristics (e.g. underrun tractor or forklift AGV) in a uniform language with a master control. This creates the basis for operating any combination of AGVs in a master control. The master control provides a contract award and coordination of the AGV traffic (cf. Chapter 5)

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 through production supply to goods out) to control free navigating vehicles and guided vehicles.

In contrast to automated vehicles, autonomous vehicles solve problems that occur on the basis of the corresponding sensor system and algorithms independently and can react accordingly to deviations in a dynamic environment or be adapted to them shortly afterwards. Autonomous properties such as the independent bypassing of obstacles can 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 expandable structure; a detailed specification required for this is not part of this document.

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

3.1 Other applicable documents

Dokument (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 decisions, e.g. with regard to the selection of routes or the behaviour at intersections

Vehicles should transfer their status immediately 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 is defined as the version to be used. Use of MQTT 5 is possible once this is redistributed and tested. MQTT allows the distribution of messages to subchannels, which are called "topics" in English. 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.

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, manoeuvrability, etc.)
- Detection and resolution of blockages "deadlocks")
- Energy management: Load 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 coordinated
- 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 AGV carrying a load and for AGV that do 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 AGVs. 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 a public Git repository (<https://github.com/MaximilianRies/vd-m-a-5050>). The branch to this document version is 5.00. The JSON schemes are the "some point of truth" and are considered a universal data set with the claim to correctness that can be relied upon as a developer of the interface

6.1 Symbols of 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 [ENG]
standard	Variable is an elementary data type
bold	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 AGVs plan their trajectory themselves and the field trajectory within the edges 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.

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 AGVs memory is insufficient to process an incoming order, it is to reject the order.

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

6.1.3 Notation of enumerations

Enumerations are basically in large letters. 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 (1, 0 or,ä.).

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.

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 is to be used for all communications.

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 (black) has to be adapted individually to match the topics defined in this protocol (red). 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/manufacture/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, prepended with a "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

6.4 Protocol Header

Each JSON starts with a header. In the following sections, the header will be referenced as header but only 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 (ISO8601, UTC); YYYY-MM-DDTHH:mm:ss.sZ (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 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 (master control) and the AGVs

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 actions that are to be executed immediately	mandatory	instantActions.schema
state	AGV	master control	Communication of the AGVs 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 a MQTT protocol level check of connection.	mandatory	connection.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 are 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's 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. waiting position in front of fire door, should only have one node for all types of AGVs. However, due to the different sizes and specifications of the AGVs, it might be necessary to deviate from this standard in certain situations.

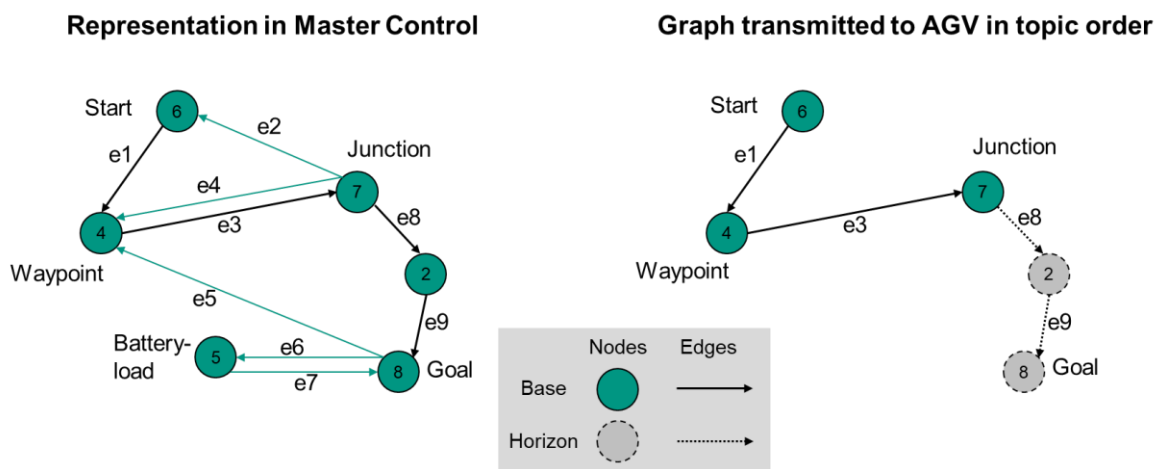


Figure 3 Graph representation in Master Control and graph transmitted in orders

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 updates

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. However, the AGV will initially only travel to the last junction of the "Base" route.

However, the AGV will initially only travel to the last junction of the "Base" route.

Since MQTT is an asynchronous protocol and transmission via WLAN 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.

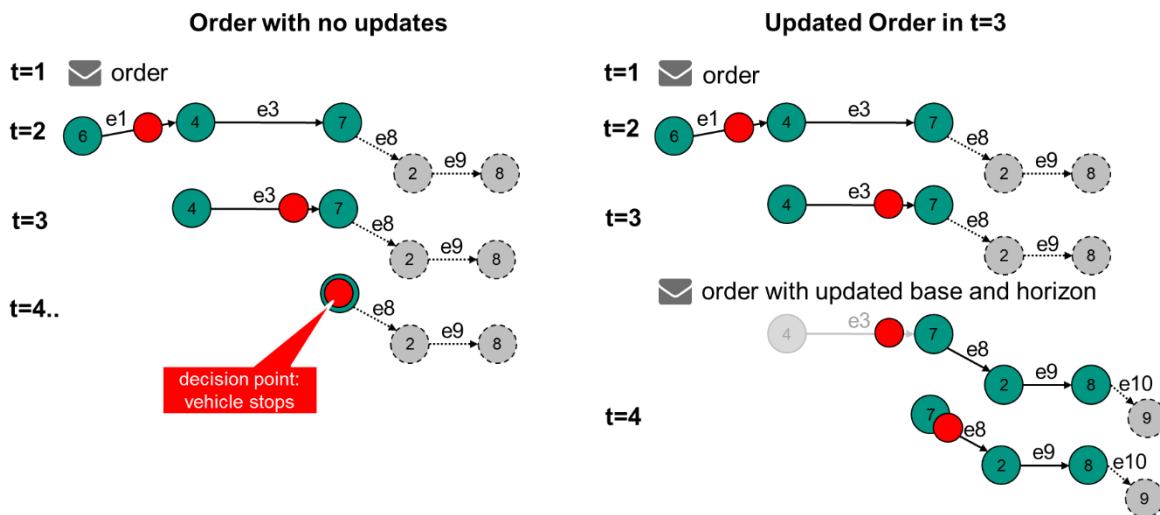


Figure 4 Procedure for changing the driving route "Horizon"

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 a 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 WIFI). 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 (Code Snippet 1, orderUpdateId 0).

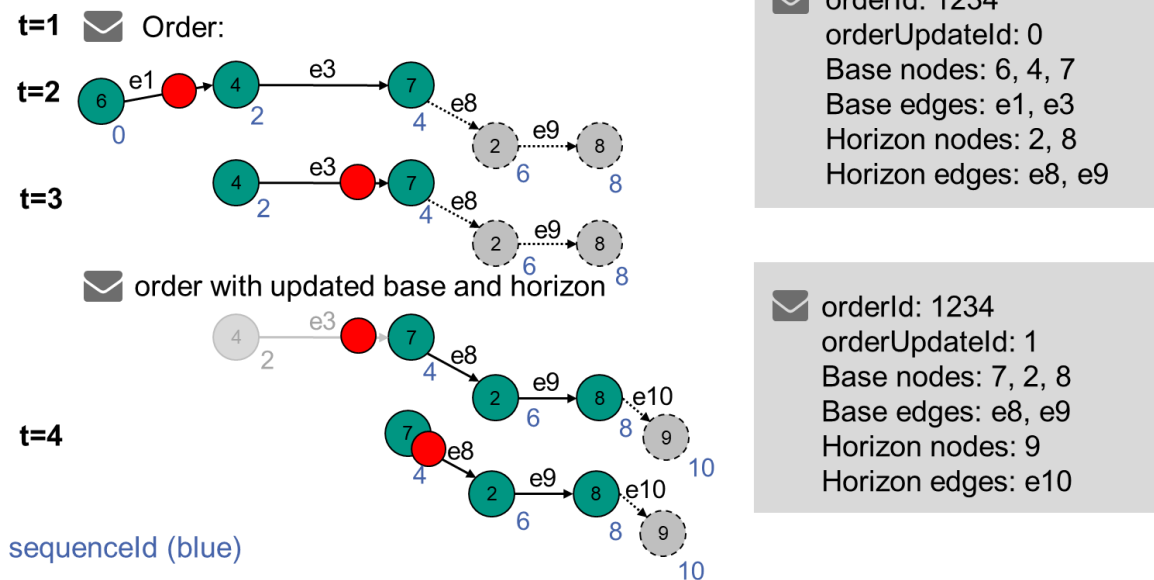


Figure 7 Regular update process - order extension

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.

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.

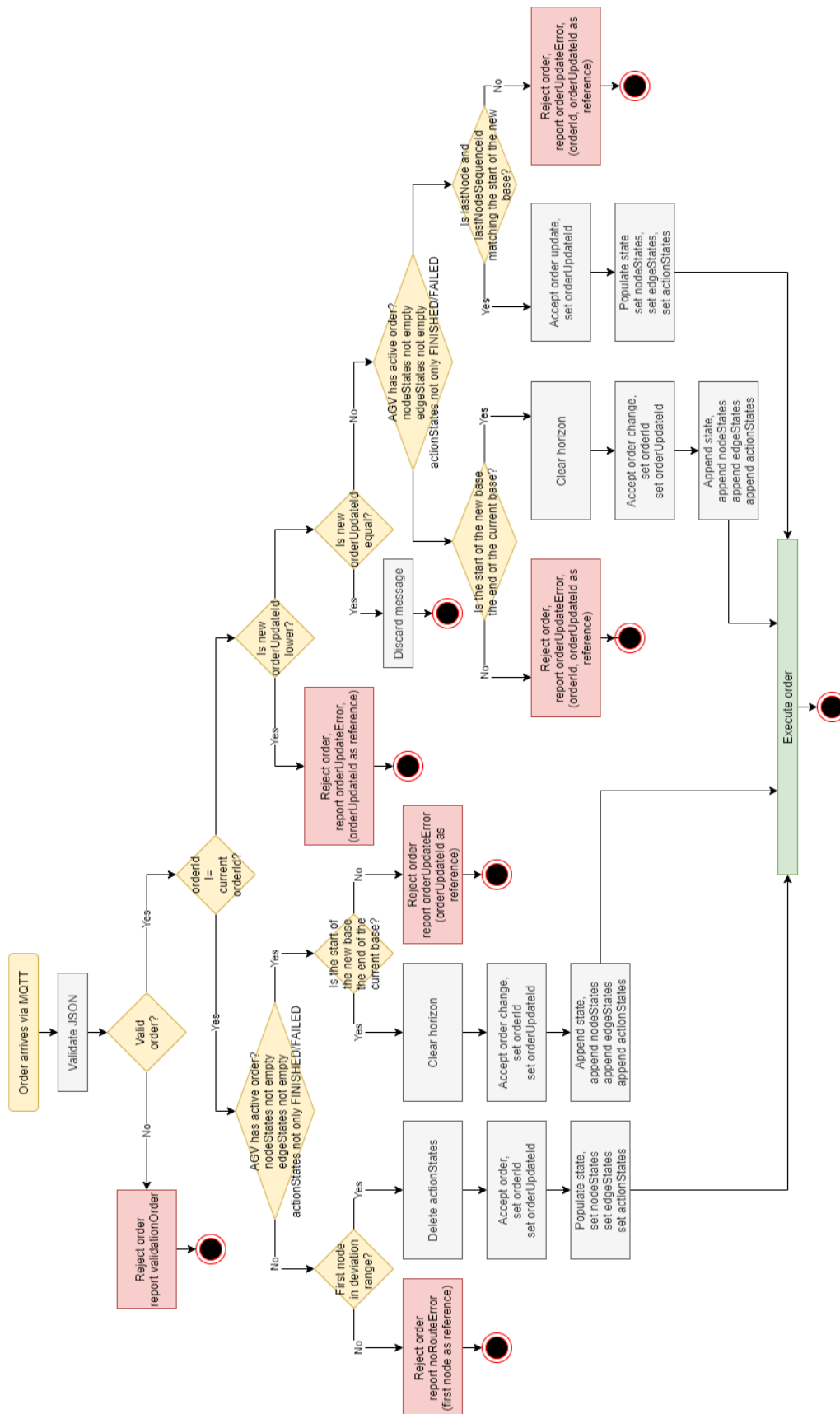


Figure 8 The process of accepting an order or orderUpdate

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, “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.

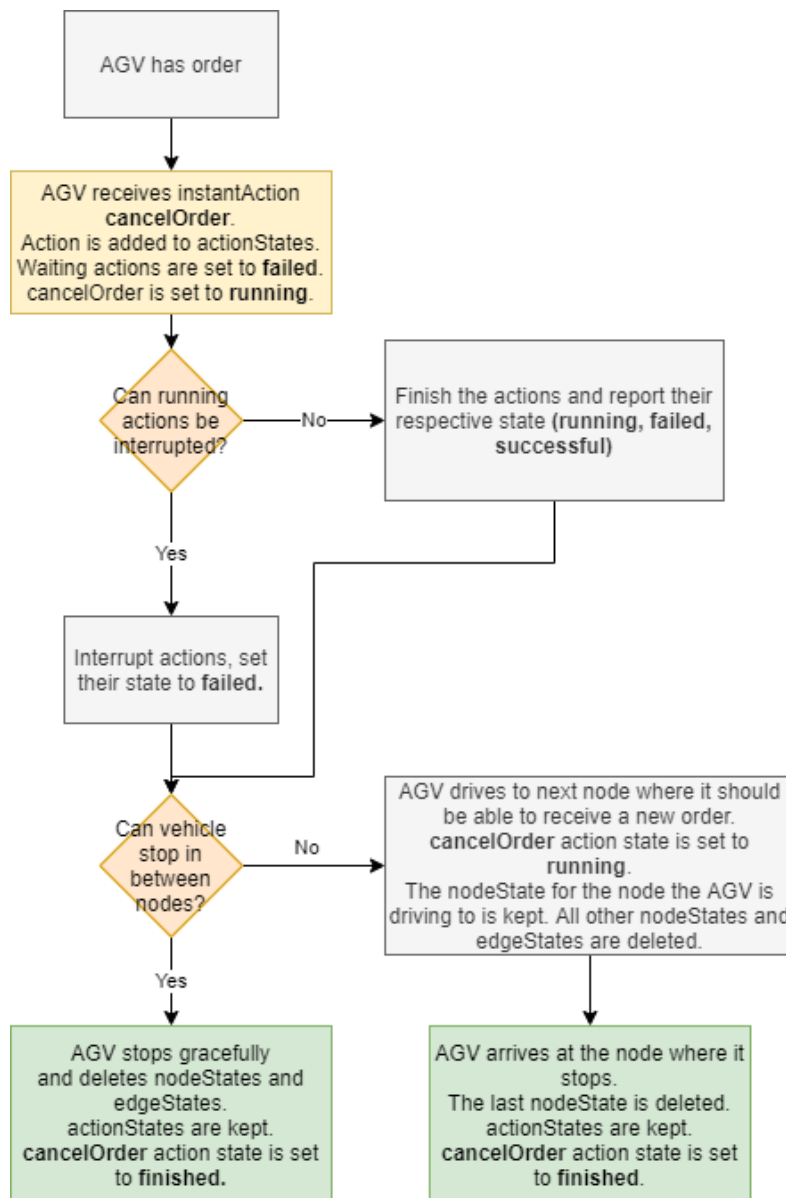


Figure 9 Expected behaviour after a `cancelOrder`

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 5).

In case of the 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 instantAction’s actionId must be passed as an errorReference.

6.6.4 Order rejection

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

6.6.4.1 Vehicle gets a malformed new order.

Resolution:

- a) Vehicle does NOT take over the new order in its internal buffer.
- b) The vehicle reports the warning “validationError”
- c) 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

- a) Vehicle does NOT take over the new order in its internal buffer
- b) Vehicle reports the warning “orderError” with the wrong fields as error references
- c) 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:

- a) Vehicle does NOT take over the new order in its internal buffer.
- b) Vehicle keeps the PREVIOUS order in its buffer.
- c) The vehicle reports the warning “orderUpdateError”
- d) 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 AGVs, 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.

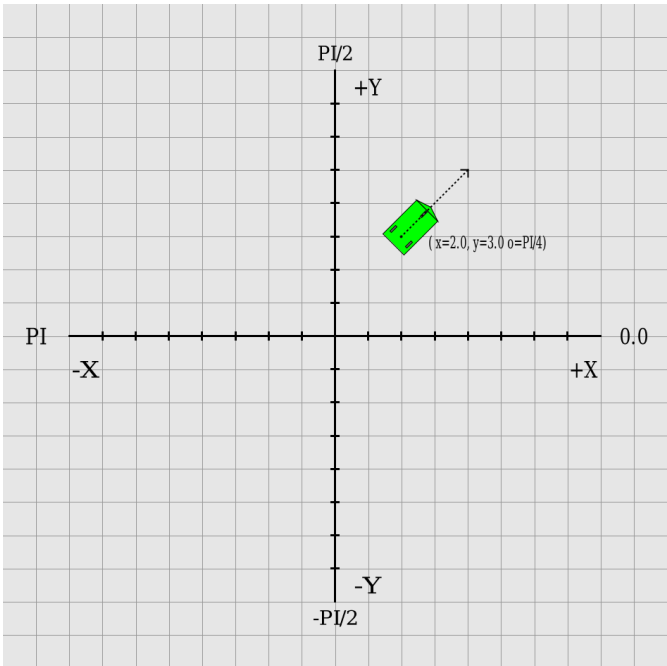


Figure 10 Coordinate system with sample agv and orientation

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

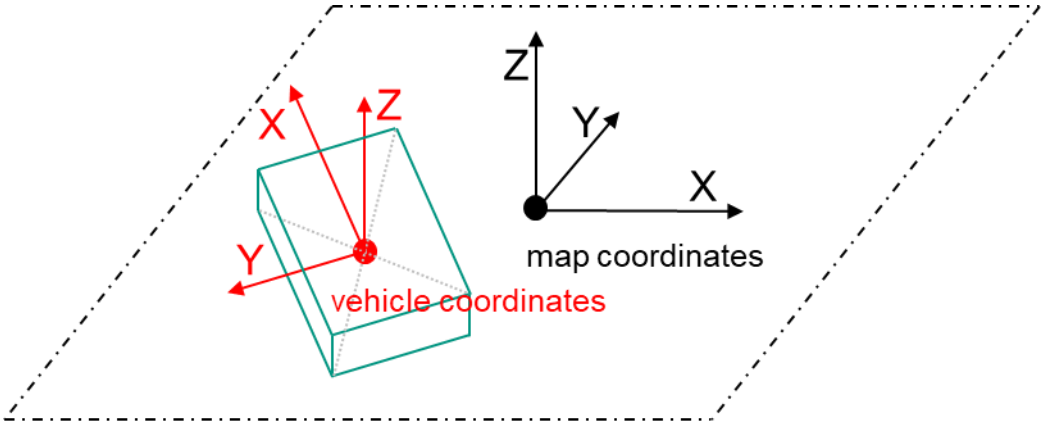


Figure 11 Coordinate systems for map and vehicle

6.7 Implementation of the order message

Object structure	Unit	Data type	Description
header		N/A	For header information see 6.4
orderId		string	Order identification. This is to be used to identify multiple order messages that belong to the same order.
orderUpdateId		Integer	orderUpdate identification. Is unique per orderId. If an order update is rejected, this field is to be passed in the rejection message
zoneSetId		String	Unique identifier of the zone set that the AGV has to use for navigation or that was used by master control for planning.

			Optional: Some master control systems do not use zones. Some AGVs do not understand zones. Do not add to message if no zones are used.
nodes [node]		array	Array of nodes objects to be traversed for fulfilling the order. One node is enough for a valid order. Leave edge list empty for that case.
edges [edge]		array	Array of edge objects to be traversed for fulfilling the order. May be empty in case only one node is used for an order.
node {		JSON-object	
nodeId		string	Unique node identification.
sequenceId		Integer	Number to track the sequence of nodes and edges in an order and to simplify order updates. The main purpose is to distinguish between a node which is passed more than once within one orderId. The variable sequenceId runs across all nodes and edges of the same order and is reset when a new orderId is issued.
nodeDescription		string	Additional information on the node
released		boolean	"true" indicates that the node is part of the base. "false" indicates that the node is part of the horizon.
nodePosition		JSON-object	Node position. Optional for vehicle-types that do not require the node position (e.g. line-guided vehicles).
actions [action]		array	Array of actions to be executed on a node. Empty array if no actions required.
}			
nodePosition {		JSON-object	Defines the position on a map in a global projectspecific 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.
theta	rad	float64	Range: [-Pi ... Pi] 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.
allowedDeviationXY		Float64	Indicates how exact an AGV has to drive over a node in order for it to count as traversed. If = 0: no deviation is allowed (no deviation means within the normal tolerance of the AGV manufacturer). If > 0: allowed deviation-radius in meters. If the AGV passes a node within the deviation-radius, the node is considered to have been traversed.

Object structure	Unit	Data type	Description
allowedDeviationTheta		Float64	Range: [0 ... Pi] 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.
mapId		string	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.
mapDescription		string	Additional information on the map
}			
action {		JSON-object	Describes an action that the AGV can perform.
actionType		string	Name of action as described in the first column of "Actions and Parameters". Identifies the function of the action.

actionId		string	Unique ID to identify the action and map them to the actionState in the state. Suggestion: Use UUIDs.
actionDescription		string	Additional information on the action
blockingType		string	Enum {NONE, SOFT, HARD} “NONE” – allows driving and other actions “SOFT” – allows other actions, but not driving “HARD” – is the only allowed action at that time.
actionParameters [actionParameter] }		array	Array of actionParameter-objects for the indicated action e. g. deviceId, loadId, external Triggers. 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.
edgeDescription		string	Additional information on the edge
released		boolean	“true” indicates that the edge is part of the base. “false” indicates that the edge is part of the horizon.
startNodeId		string	nodeId of startNode
endNodeId		string	nodeId of endNode
maxSpeed	m/s	Float64	Permitted maximum speed on the edge. Speed is defined by the fastest measurement of the vehicle.
maxHeight	m	Float64	Permitted maximum height of the vehicle, including the load, on edge
minHeight	m	Float64	Permitted minimal height of the load handling device on the edge.
orientation	rad	Float64	Orientation of the AGV on the edge relative to the global project specific map coordinate origin (for holonomic vehicles with more than one driving direction). Example: orientation Pi/2 rad will lead to a rotation of 90 degrees. 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. If a trajectory with orientation is defined, follow the trajectories orientation. If a trajectory without orientation and the orientation field here is defined, apply the orientation to the tangent of the trajectory.

Object structure	Unit	Data type	Description
direction		string	Sets direction at junctions for line-guided or wire-guided vehicles, to be defined initially (vehicle-individual). Example: left, right, straight, 433MHz
rotationAllowed		boolean	“true”: rotation is allowed on the edge. “false”: rotation is not allowed on the edge. Optional: Default to “false”.
maxRotationSpeed	rad/s	Float64	Maximum rotation speed Optional: No limit if not set.
trajectory		JSON-object	Trajectory JSON-object for this edge as a NURBS. Defines the curve on which the AGV should move between startNode and endNode. Optional: Can be omitted if AGV cannot process trajectories or if AGV plans its own trajectory.
length	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

<pre> } trajectory { degree knotVector [float64] controlPoints [controlPoint] } controlPoint { x y orientation weight } </pre>			the AGV leaves the edge, the action will stop and the state before entering the edge will be restored.
		JSON-object	
		Float64	Range: [1 ... ∞] 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.
		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.
		array	List of JSON controlPoint objects defining the control points of the NURBS, which includes the beginning and end point.
		JSON-object	
		float64	X coordinate described in the world coordinate system.
		float64	Y coordinate described in the world coordinate system.
	rad	float64	Range: [-π ... π] Orientation of the AGV on this position of the curve. The orientation is in world coordinates. When not defined the orientation of the AGV will be tangential to the curve.
		float64	Range: [0 ... ∞] 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 definitions, 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	yes	-	.paused	yes	no	no

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

general					scope			
action	counter action	Description	Important	Para-meter	linked state	instant	node	edge
initPosition	-	Resets (overrides) the pose of the AGV with the given parameters.	yes	x (float64) y (float64) theta (float64) mapId (String) lastNodeID (String)	.agvPosition.x .agvPosition.y .agvPosition.theta .agvPosition.mapId .agvPosition.lastNodeID	yes	yes (Elevator)	no
stateRequest	-	Requests the AGV to send a new state report	yes		-	yes	no	no
logReport	-	Request the AGV to generate and store a log report.	yes	reason (String)	-	yes	no	no
pick	drop (sofern automatisiert)	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

general					scope			
action	counter action	Description	Important	Para-meter	linked state	instant	node	edge
drop	pick (sofern automatisiert)	Request the AGV to drop a load. See action pick for more details.	no	lhd (String, optional) stationType (String, optional) stationName (String, optional) loadType (String, optional) loadId (String, optional) height (float64, optional) depth (float64, optional)load	no	yes	yes
detectObject	-	AGV detects object (e.g. load, charging spot, free parking position).	yes	objectType(String, optional)	-	no	yes	yes
finePositioning	-	On a node, AGV will position exactly on a target. The AGV is allowed to deviate from its node position. On an edge, AGV will e.g. align on stationary equipment while traversing an edge. InstantAction: AGV starts positioning exactly on a target.	yes	stationType(String, optional) stationName(String, optional)	-	no	yes	yes
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	yes	-	-	yes	no	no

		order is deleted. All actions are canceled.						
--	--	---	--	--	--	--	--	--

6.8.2 Predefined action definitions, description of their states

action	action states				
	initializing	running	paused	finished	failed
startPause	-	Activation of the mode is in preperation. 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 preperation. 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. overridden by hardware switch).
startCharging	-	Activation 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 started. The AGV reports .batteryState.charging: true.	The charging process could not be started for some reason (e.g. not aligned to charger). Charging problems should correspond with an error.
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.

action	action states				
	initializing	running	paused	finished	failed
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.
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 was 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.	-

6.9 Topic: instantActions (from master 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
<i>header</i>	N/A	For header information see 6.4.
instantActions [action]	array	Array of actions that need to be performed immediately and are not part of the regular order.

When an AGV receives an instantAction, an appropriate actionStatus is added to the actionStates array of the AGV state. The actionStatus is updated according to the progress of the action. See also Figure 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 AGVs 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 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/edgeStates” includes all nodes/edges that the AGV still must traverse.

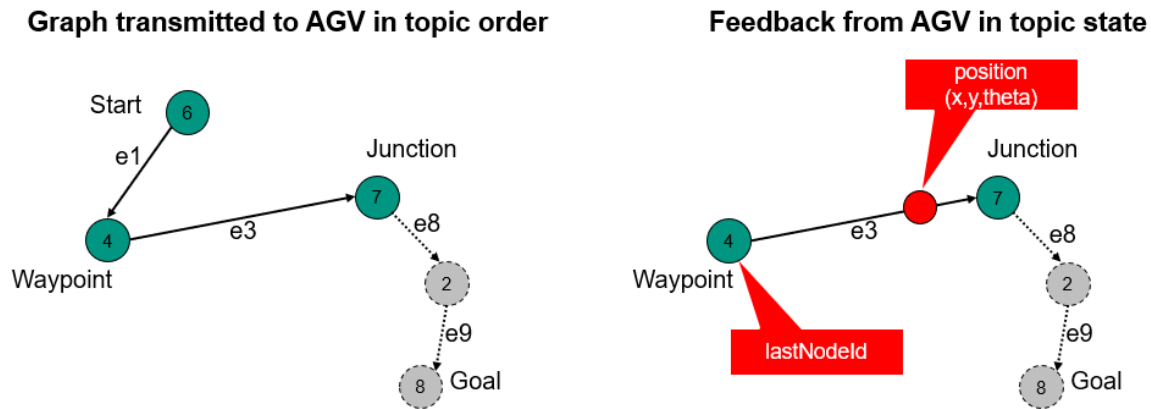


Figure 12 Order Information provided by the state topic. Only the ID of the last node and the remaining nodes and edges are transmitted

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.

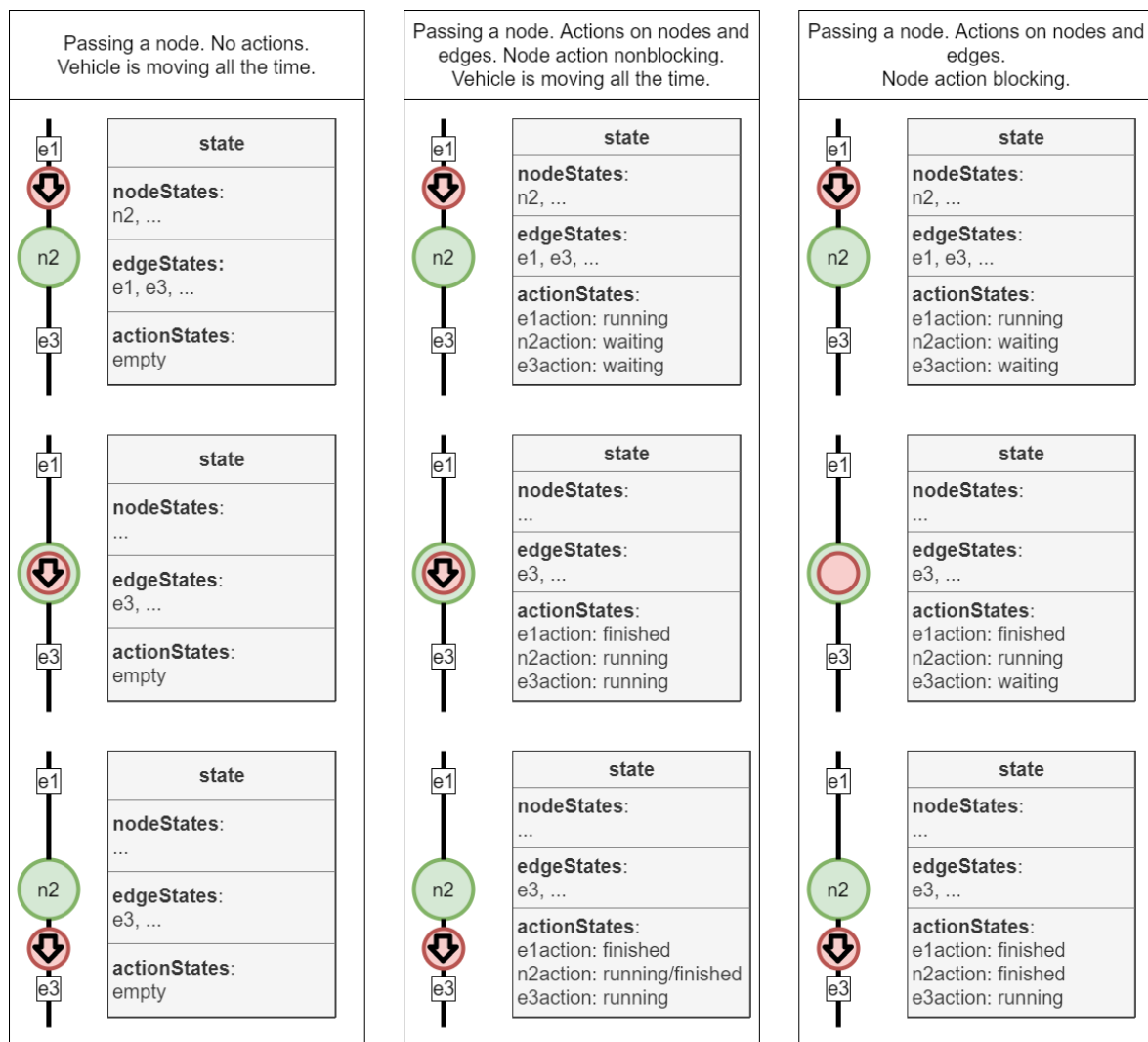


Figure 13 nodeStates, edgeStates, actionStates during order handling

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 visualisation and debugging purposes.

6.10.5 Errors

The AGV reports errors via the errors array. Errors have to 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
header		N/A	For header information see 6.4
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.
zoneSetId		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	nodeId 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	sequenceId of the last reached node or, if the AGV is currently on a node, sequenceId 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)
agvPosition		JSON-object	Current position of the AGV on the map. Optional: Can only be omitted for AGVs without the capability to localize themselves, e.g. line guided AGVs.
velocity		JSON-object	The AGVs velocity in vehicle coordinates.
loads [load]		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.
driving		boolean	“true”: indicates that the AGV is driving and/or rotating. Other movements of the AGV (e.g. lift movements) are not included here. “false”: indicates that the AGV is neither driving nor rotating.
paused		Boolean	True: AGV is currently in a paused state, either because of the push of a physical button on the AGV or because of an instantAction. The AGV can resume the order. False: The AGV is currently not in a paused state.

Object structure	Unit	Data type	Description
newBaseRequest		Bool	“true”: AGV is almost at the end of the base and will reduce speed if no new base is transmitted. Trigger for master control to send a new base. “false”: no base update required.
distanceSinceLastNode	meter	float64	Used by line guided vehicles to indicate the distance it has been driving past the „lastNodeId“. Distance is in meters.

actionStates [actionState]		array	Contains a list of the current actions and the actions which are yet to be finished. This may include actions from previous nodes that are still in progress. When an action is completed, an updated state message is published with actionStatus set to finished and if applicable with the corresponding resultDescription. The action state is kept until a new order is received.
batteryState		JSON-object	Contains all battery-related information.
operatingMode		string	Enum {AUTOMATIC, SEMIAUTOMATIC, MANUAL, SERVICE, TEACHIN} For additional information, see the table OperatingModes in the chapter 6.10.6.
errors [error]		array	Array of error-objects. All active errors of the AGV should be in the list. An empty array indicates that the AGV has no active errors.
informations [info]		Array	Array of info-objects. An empty array indicates that the AGV has no information. This should only be used for visualization or debugging – it must not be used for logic in master control.
safetyState		JSON-object	Contains all safety-related information.
nodeState {		JSON-object	
nodeId		string	Unique node identification
sequenceId		integer	sequenceId to discern multiple nodes with same nodeId.
nodeDescription		string	Additional information on the node
nodePosition		JSON-object	Node position. The object is defined in chapter 6.6 Optional: master control has this information. Can be sent additionally, e. g. for debugging purposes.
released		bool	“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		integer	sequenceId to differentiate between multiple edges with the same edgeId
edgeDescription		string	Additional information on the edge
released		boolean	“true” indicates that the edge is part of the base. “false” indicates that the edge is part of the horizon.
trajectory		JSON-object	The trajectory is to be communicated as a NURBS and is defined in chapter 6.4 Trajectory segments are from the point where the AGV starts to enter the edge until 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	“false”: position is not initialized “true”: position is initialized

Object structure	Unit	Data type	Description
<i>localizationScore</i>		float64	Range: [0.0 ... 1.0] Describes the quality of the localization and therefore, can be used e. g. by SLAM-AGVs 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.
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.
mapDescription		string	Additional information on the map.
velocity {		JSON-object	
vx	m/s	Float64	The AGVs velocity in its x direction
vy	m/s	Float64	The AGVs velocity in its y direction
omega	Rad/s	Float64	The AGVs turning speed around its z axis
}			
load {		JSON-object	
loadId		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.
loadType		string	Type of load
loadPosition		string	Indicates which load handling/carrying unit of the AGV is used, e. g. in case the AGV has multiple spots/positions to carry loads. For example: "front", "back", "positionC1", etc. Optional for vehicles with only one loadPosition
boundingBoxReference		JSON-object	Point of reference for the location of the bounding box. The point of reference is always the center of the bounding box's bottom surface (at height = 0) and is described in coordinates of the AGV's coordinate system.
loadDimensions		JSON-object	Dimensions of the load's bounding box in meters.
weight	kg	uint32	Range: [0.0 ... ∞) Absolute weight of the load measured in kg.
}			

Object structure	Unit	Data type	Description
boundingBoxReference {		JSON-object	Point of reference for the location of the bounding box. The point of reference is always the center of the bounding box's bottom surface (at height = 0) and is described in coordinates of the AGV's coordinate system.
x		float64	x-coordinate of the point of reference.
y		float64	y-coordinate of the point of reference.
z		float64	z-coordinate of the point of reference.
theta		float64	Orientation of the loads bounding box. Important for tugger trains etc.
}			
loadDimensions {		JSON-object	Dimensions of the load's bounding box in meters.

length	m	float64	Absolute length of the load's bounding box.
width	m	float64	Absolute width of the load's bounding box.
height	m	float64	Absolute height of the load's bounding box. Optional: Set value only if known.
}			
actionState {		JSON-object	
actionId		string	action_ID
actionType		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.
actionStatus		string	Enum {waiting; initializing; running; paused; finished; failed} waiting: waiting for trigger (passing the node, entering the edge) Paused: paused by instantAction or external trigger failed: action could not be performed.
<i>resultDescription</i>		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).
<i>batteryVoltage</i>	V	float64	Battery Voltage

Object structure	Unit	Data type	Description
<i>batteryHealth</i>	%	int8	Range: [0...100] State of Health
charging		boolean	"true": charging in progress "false": AGV is currently not charging
reach	m	uint32	Range: [0.0 ... ∞) Estimated reach with current State of Charge
}			
error {		JSON-object	
errorType		string	Type / name of error
<i>errorReferences</i> <i>[errorReference]</i>		array	Array of references to identify the source of the error (e. g. headerId, orderId, actionId, ...). For additional information see „Praktische Beispiele“ chapter 7

<i>errorDescription</i>		string	Error description
<i>errorLevel</i>		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	
<i>referenceKey</i>		string	References the type of reference (e. g. headerId, orderId, actionId, ...).
<i>referenceValue</i>		string	References the value, which belongs to the reference key.
}			
info {		JSON-object	
<i>infoType</i>		string	Type / name of information
<i>infoReferences</i> <i>[infoReference]</i>		array	Array of references
<i>infoDescription</i>		string	Info description
<i>infoLevel</i>		string	Enum {DEBUG, INFO} DEBUG: used for debugging INFO: used for visualization
}			
safetyState {		JSON-object	
<i>eStop</i>		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
<i>fieldViolation</i>		boolean	Protective field violation. "true": field is violated "false": field is not violated
}			

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

Identifier	Data type	Type	Description
operatingMode	string	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 via their field “actionStatus” at which stage of the actions lifecycle the action is.

Table 1 describes, which value the enum actionStatus can hold

actionStatus	Description
WAITING	Action was received by AGV but the node where it triggers was not yet reached or the edge where it is active was not yet entered.
INITIALIZING	Action was triggered, preparatory measures are initiated.
RUNNING	The action is running.
PAUSED	The action is paused because of a pause instantAction or external trigger (pause button on AGV)
FINISHED	The action is finished. A result is reported via the resultDescription
FAILED	Action could not be finished for whatever reason.

Table 1 The acceptable values for the actionStatus field

A state transition diagram is provided in Figure 10.

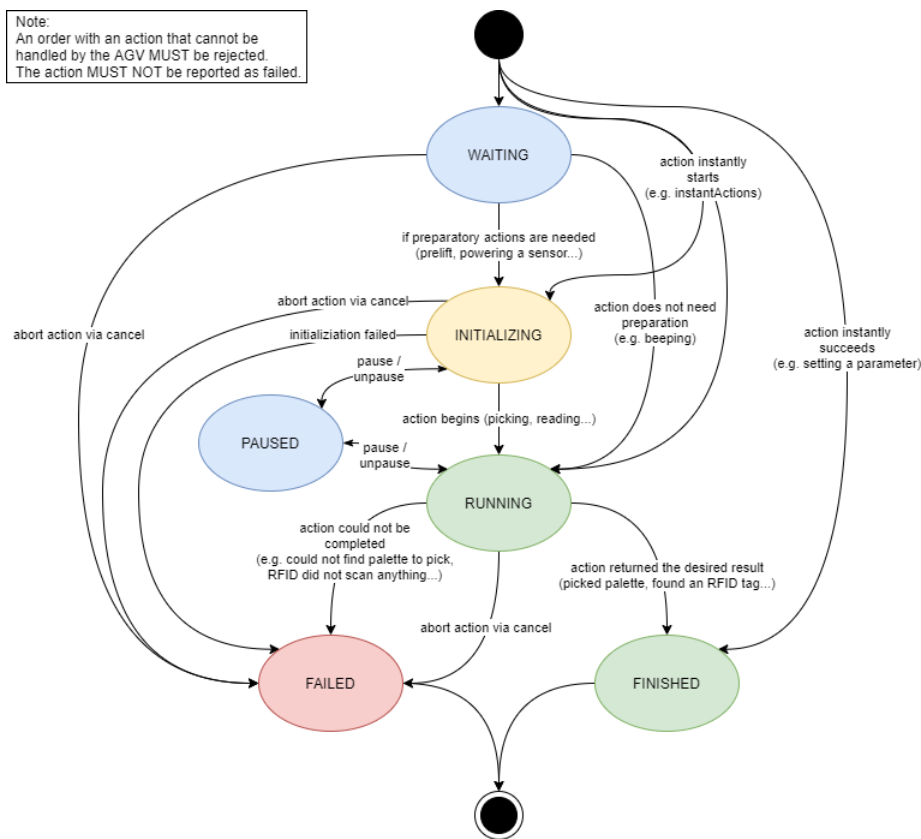


Figure 14 All possible status transitions for actionStates

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 11 describes how the AGV should handle these actions.

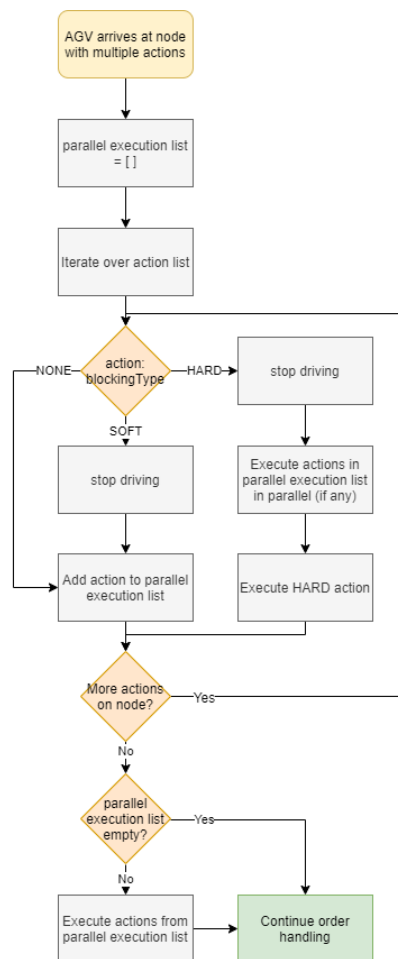


Figure 15 Handling multiple actions

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 AGVs client from the broker. Thus, the master control can detect a disconnection event by subscribing the connection topics of all AGVs. 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 suggested last will topic structure is:

uagv/v2/manufacture/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{ONLINE, OFFLINE, CONNECTIONBROKEN} ONLINE: connection between AGV and broker is active. OFFLINE: connection between AGV and broker has gone offline in a coordinated way. CONNECTIONBROKEN: The connection between AGV and broker has unexpectedly ended.

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

AGV wants to disconnect gracefully:

1. AGV sends „uagv/v2/manufacturer/SN/connection“ with connectionState = „offline“.
2. Disconnect the mqtt connection with a disconnect command.

AGV comes online:

1. Set the lastwill to „uagv/v2/manufacturer/SN/connection“ with the field „connectionState“ = „connectionBroken“, when the mqtt connection is created.
2. Send the topic „uagv/v2/manufacturer/SN/connection“ with connectionState = „online“.

All messages of this topic should be send 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“.

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 5.6.2 Implementation). 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

8.1 Definition

Concept	Description
Free navigation AGVs	<p>Vehicles that use a map to plan their own path. 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 (physically or virtual)	<p>Vehicles that get their path sent by the master control. 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. This is initially created and then used. A future version of the interface will make it possible to transfer this card to the vehicles (e.g. for free navigation).</p>