VDA 5050
Implementing a universal TypeScript/JavaScript library

Software Architecture & Design
Usage examples

SIEMENS

VDA 5050 – Specification for Driverless Transport Systems (DTS)

Interface for the communication between AGVs and master control

Overview

- Joint project team and standardization effort driven by VDA and VDMA*
- Current specification versions 1.1 and 2.0

Objectives

- Create a universally applicable and uniform interface to coordinate *automated guided vehicles* (AGV) by a *master control* (e.g. fleet manager)
- Enable parallel operation with AGVs from different manufacturers and conventional systems in the same work environment
- Reduce complexity and increase plug-and-play capabilities across transport vehicles, vehicle models, and manufacturers
- Increase manufacturer's independence using common interfaces between vehicle control and coordination level

** VDA - German Association of the Automotive Industry, VDMA Materials Handling and Intralogistics Association*

SIEMENS

VDA 5050 – Specification for Driverless Transport Systems (DTS)*

Interface for the communication between AGVs and a master control

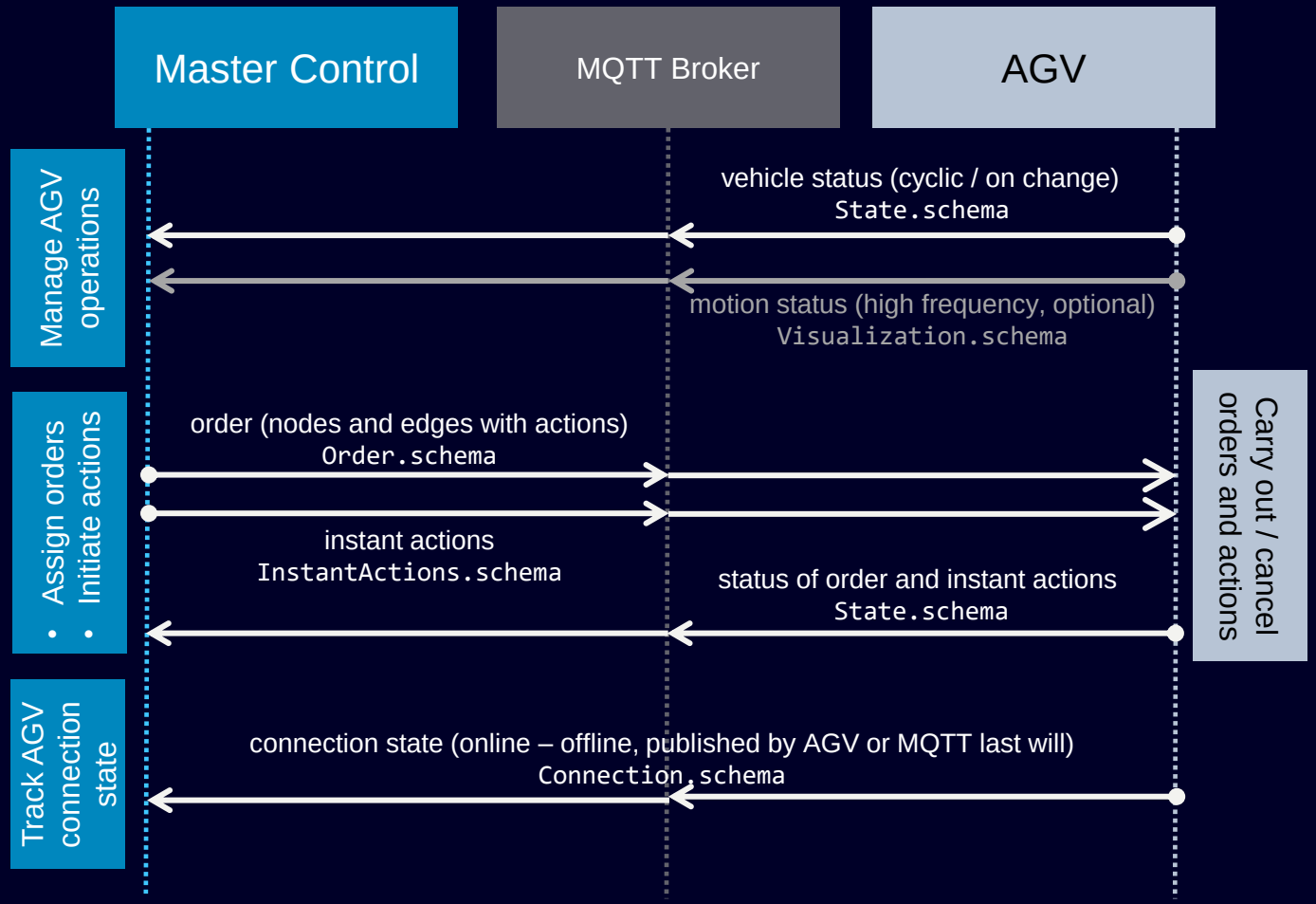
Master Control for AGVs

coordination plane

*Communication interface** uses MQTT, a broker-based pub-sub messaging protocol with JSON payload format*

Automated Guided Vehicles

vehicle plane



** According to VDA 5050 Specification Version 1.1*

*** Designed to support a minimum of 1000 vehicles with different degrees of autonomy*

SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library
Motivation

VDA 5050 specification characteristics

- Overly rich and opinionated feature set, encompassing all types of AGV (e.g. line-guided, free navigation) and satisfying dedicated requirements of AGV users and manufacturers involved
- Complex control logic and information flows on coordination and vehicle plane
- No reference implementation yet; huge effort to implement one

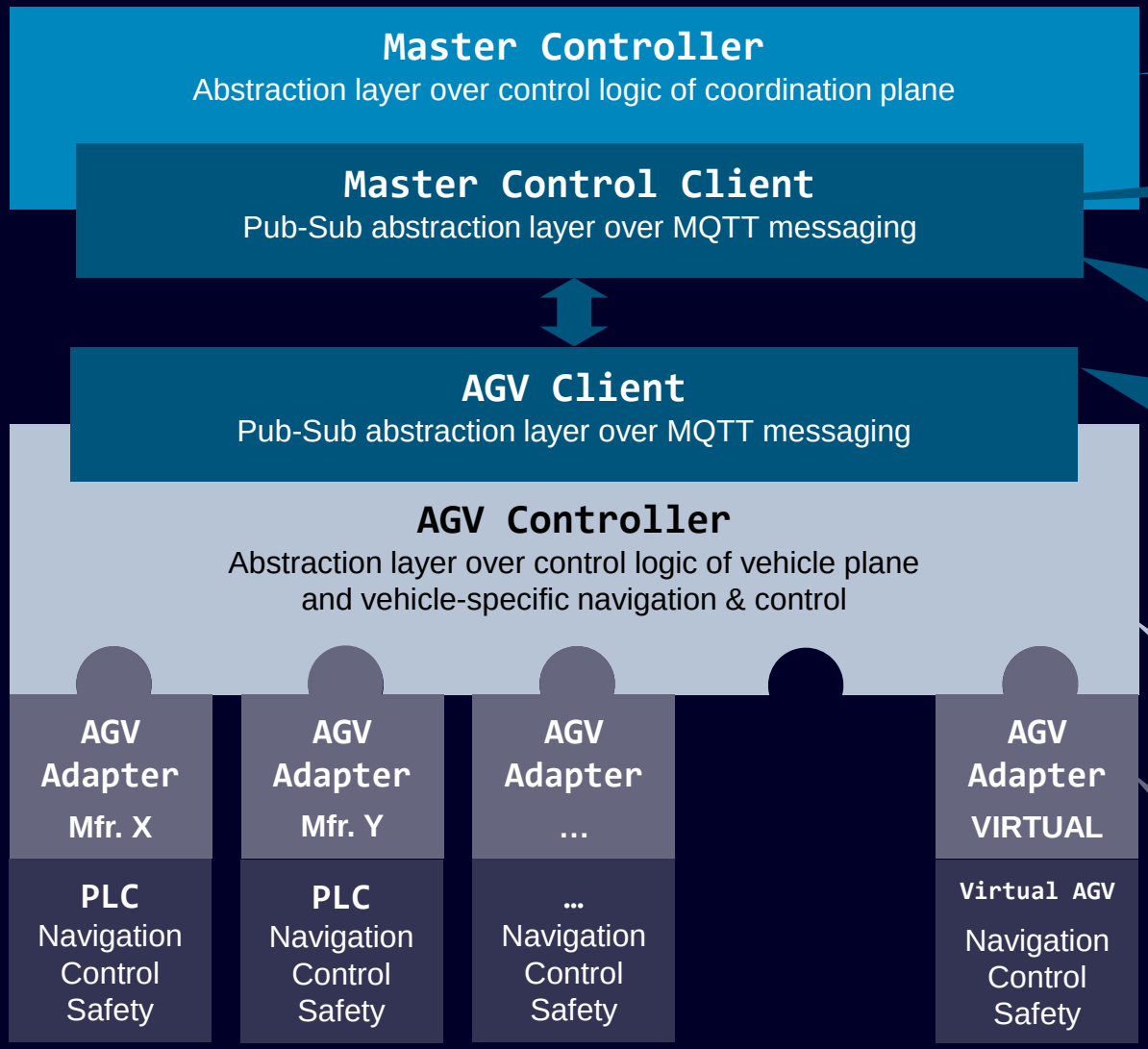
Key objectives of library

- Encapsulate complex control logic into reusable and interoperable components
- Provide abstraction layers for coordination/vehicle plane and communication
- Provide uniform protocol to adapt to vehicle-specific navigation & control interfaces
- Use library components on coordination/vehicle plane in combination or separately
- Support custom VDA 5050 actions and extension topics/object models

SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library

Software architecture with configurable, extensible, and pluggable components



- ➢ *Assign orders and initiate instant actions to AGVs*
- ➢ *Report order/instant action execution state to application*

- ➢ *Target individual, all, or a subset of AGVs within the DTS*
- ➢ *Track AGV connection state*

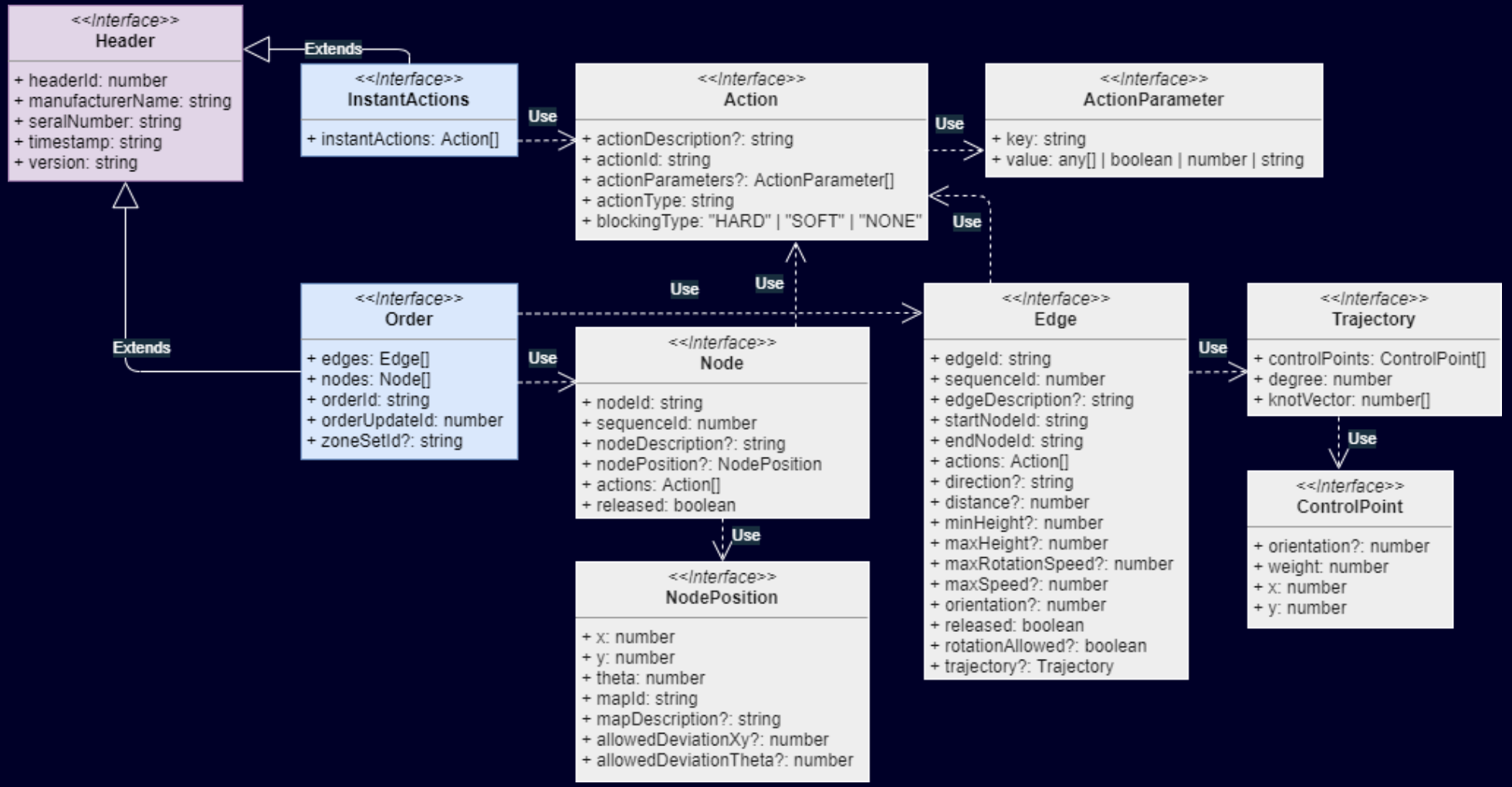
- ✓ *Autogenerate VDA 5050 object types from JSON schemas*
- ✓ *Autogenerate MQTT topic-payload from VDA 5050 objects*
- ✓ *Configurable MQTT protocol version (3.1.1, 5.0)*
- ✓ *Configurable MQTT transport options: TCP, WebSocket, TLS*
- ✓ *Configurable MQTT topic structure to support Edge brokers*
- ✓ *Configurable automatic reconnect*
- ✓ *Configurable regular heartbeat exchange with MQTT broker*
- ✓ *Offline buffering of publications and subscriptions*
- ✓ *Validation of inbound and outbound VDA 5050 messages*
- ✓ *Efficient subscription-based message dispatching*
- ✓ *Supports all standard VDA 5050 topics and schemas*
- ✓ *Extensible by custom VDA 5050 topics and schemas*

- ➢ *Process orders and instant actions according to VDA 5050 logic*
- ➢ *Manage & report vehicle/order/action state to master controller*
- ➢ *Provide uniform AGV interface for plug-in adapters*

- ➢ *Traverse edge; Stop traversing edge*
- ➢ *Execute, terminate, cancel a node, edge, or instant action*
- ➢ *Report changes in action & vehicle state to AGV controller*
- ➢ *Check if route traversable, action executable, AGV within deviation*
- ➢ *Calculate trajectory paths (if required)*

Universal VDA 5050 TypeScript/JavaScript Library

VDA 5050 object types*: InstantActions, Order

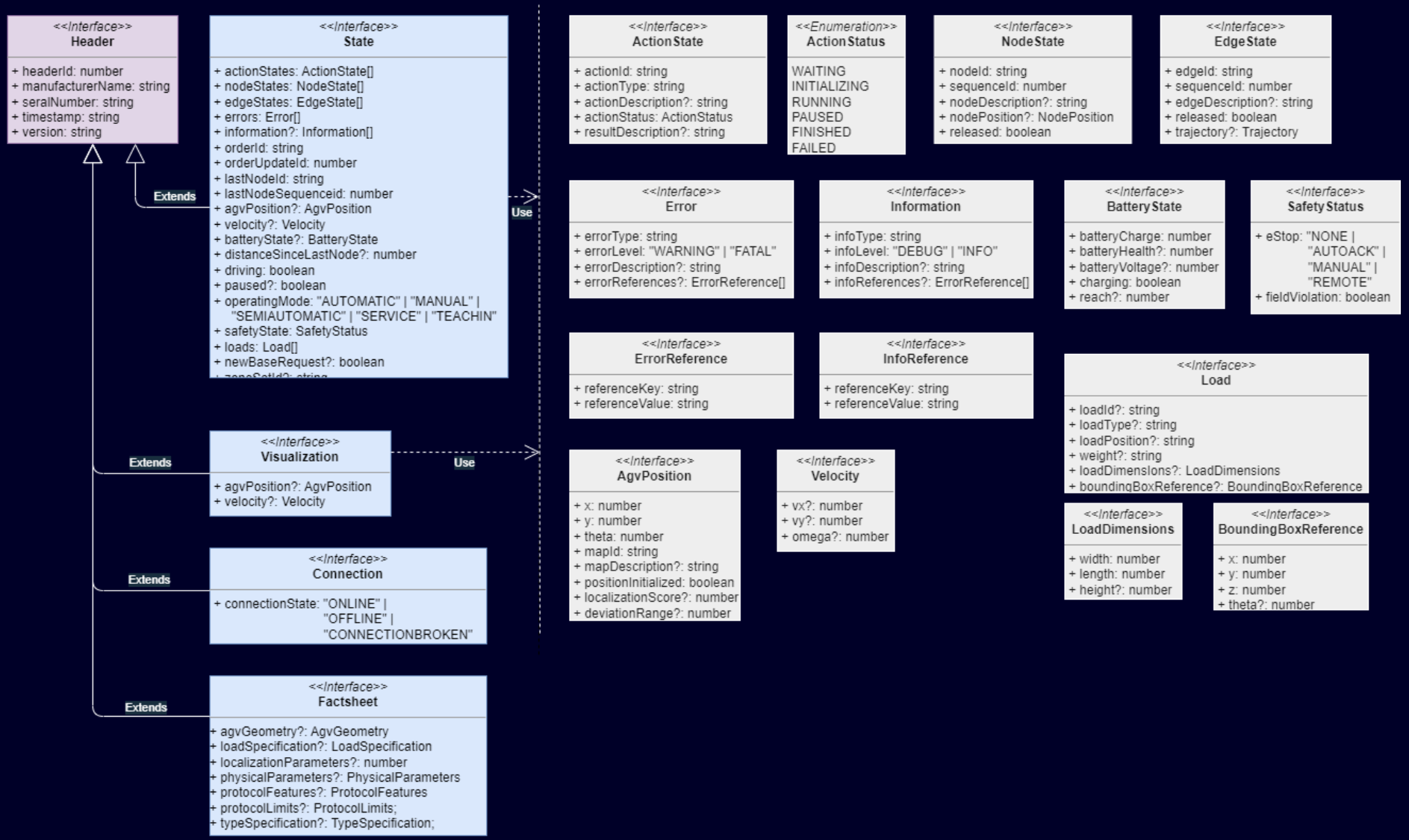


** VDA 5050 object types for TypeScript are automatically generated from VDA 5050 JSON schemas using npm package [vda-5050-cli](vda-5050-cli)*

SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library

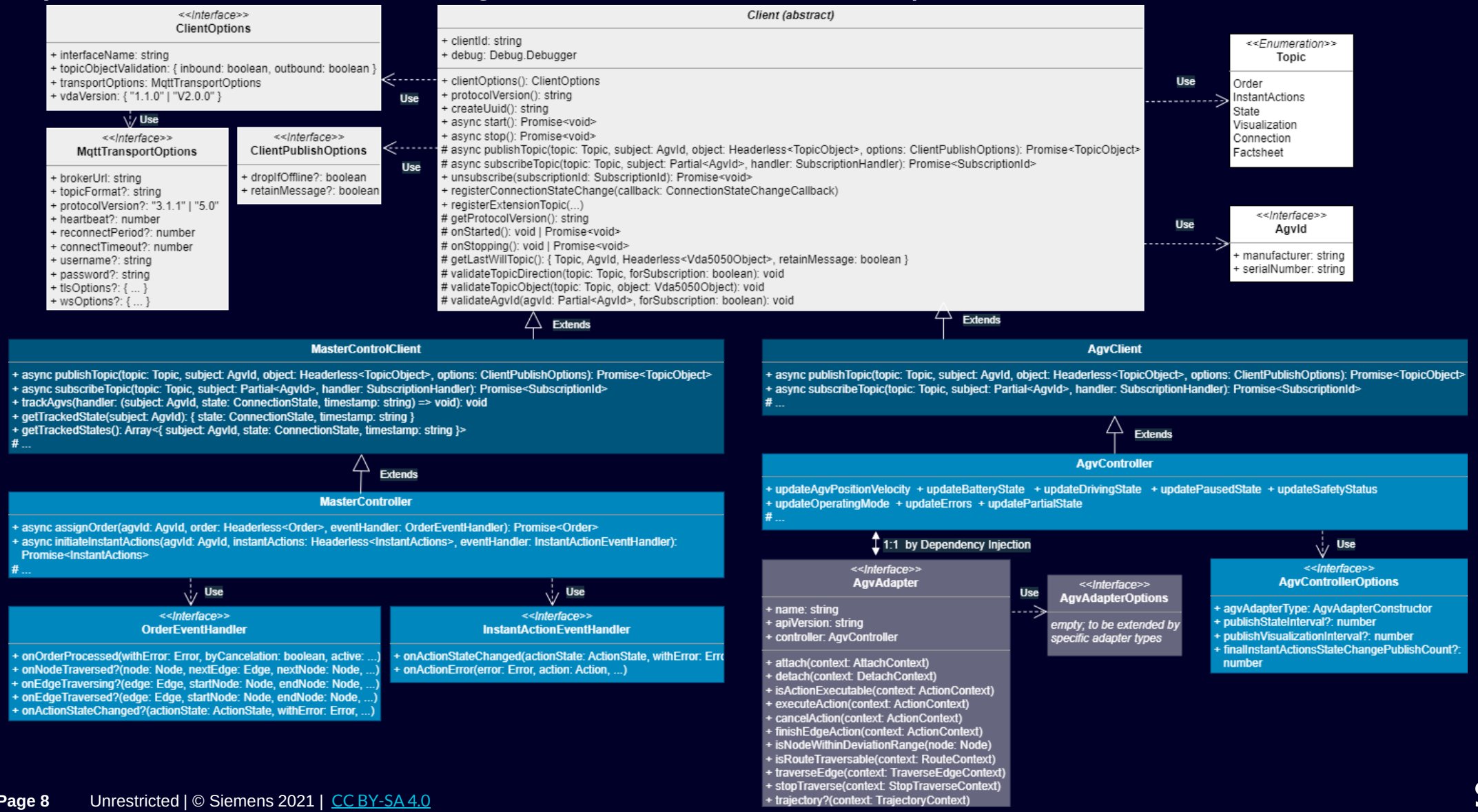
VDA 5050 object types: State, Visualization, Connection, Factsheet (V2.0)



SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library

Object-oriented software design – Client, Controller, Adapter classes



SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library
Usage example – Control Logic Abstraction Layer – Master Controller

```
// Create instance of Master Controller with minimal client options: communication namespace and broker endpoint address.
const masterController = new MasterController({ interfaceName: "logctx42", transport: { brokerUrl: "mqtt://mybroker.com:1883" }, vdaVersion: "2.0.0" }, {});

// The target AGV.
const agvId001: AgvId = { manufacturer: "RobotCompany", serialNumber: "001" };

// Define a pick & drop order with two base nodes and one base edge.
const order: Headerless<Order> = {
    orderId: masterController.createUuid(),
    orderUpdateId: 0,
    nodes: [
        {
            nodeId: "productionunit_1", sequenceId: 0, released: true, nodePosition: { x: 0, y: 0, mapId: "local" },
            actions: [{ actionId: "a001", actionType: "pick", blockingType: BlockingType.Hard, actionParameters: [{ key: "stationType", value: "floor" }, { key: "loadType", value: "EPAL
        },
        {
            nodeId: "productionunit_2", sequenceId: 2, released: true, nodePosition: { x: 100, y: 200, mapId: "local" },
            actions: [{ actionId: "a002", actionType: "drop", blockingType: BlockingType.Hard, actionParameters: [{ key: "stationType", value: "floor" }, { key: "loadType", value: "EPAL
        },
    ],
    edges: [
        { edgeId: "productionunit_1_2", sequenceId: 1, startNodeId: "productionunit_1", endNodeId: "productionunit_2", released: true, actions: [] },
    ],
};

// Start client interaction, connect to MQTT broker.
await masterController.start();

// Assign order to target AGV and handle incoming order change events.
await masterController.assignOrder(agvId001, order, {
    onOrderProcessed: (withError, byCancelation, active, ctx) => console.log("Order processed"),

    // Optional callbacks, use if required.
    onNodeTraversed: (node, nextEdge, nextNode, ctx) => console.log("Order node traversed: %o", node),
    onEdgeTraversing: (target, ctx) => console.log("Order action state changed: %o %o %o", actionState, action, target),
});
```


SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library
Usage example – Control Logic Abstraction Layer – AGV Controller

```
// Use minimal client options: communication namespace and broker endpoint address.
const agvClientOptions: ClientOptions = { interfaceName: "logctx42", transport: { brokerUrl: "mqtt://mybroker.com:1883" }, vdaVersion: "2.0.0" };

// The target AGV.
const agvId001: AgvId = { manufacturer: "RobotCompany", serialNumber: "001" };

// Specify associated adapter type; use defaults for all other AGV controller options.
const agvControllerOptions = {
    agvAdapterType: VirtualAgvAdapter,
};

// Use defaults for all adapter options of Virtual AGV adapter.
const agvAdapterOptions: VirtualAgvAdapterOptions = {};

// Create instance of AGV Controller with client, controller, and adapter options.
const agvController = new AgvController(agvId001, agvClientOptions, agvControllerOptions, agvAdapterOptions);

// Start client interaction, connect to MQTT broker.
await agvController.start();
```


SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library
Usage example – Pub-Sub Abstraction Layer – Master Control Client

```
// Create instance of Master Control Client with minimal options: communication namespace and broker endpoint address.
const mcClient = new MasterControlClient({ interfaceName: "logctx42", transport: { brokerUrl: "mqtt://mybroker.com:1883" }, vdaVersion: "2.0.0" });

// Start client interaction, connect to MQTT broker.
await mcClient.start();

// Observe Visualization objects from all AGVs manufactured by "RobotCompany".
const visSubscriptionId = await mcClient.subscribe(Topic.Visualization, { manufacturer: "RobotCompany" }, vis => console.log("Visualization object received: %o", vis));

// Publish an Order object targeted at a specific AGV.
const agvId001: AgvId = { manufacturer: "RobotCompany", serialNumber: "001" };
const order: Headerless<Order> = {
    orderId: "order0001",
    orderUpdateId: 0,
    nodes: [{ nodeId: "productionunit_1", sequenceId: 0, released: true, actions: [] }, { nodeId: "productionunit_2", sequenceId: 2, released: true, actions: [] }],
    edges: [{ edgeId: "edge1_1", sequenceId: 1, startNodeId: "productionunit_1", endNodeId: "productionunit_2", released: true, actions: [] }],
};
const orderWithHeader = await mcClient.publish(Topic.Order, agvId001, order);
console.log("Published order %o", orderWithHeader);

// Observe State objects emitted by the specific AGV Client.
const stateSubscriptionId = await mcClient.subscribe(Topic.State, agvId001, state => {
    console.log("State object received: %o", state);
    // Detect order state changes by delta comparison of received State objects.
});

// Track online-offline connection state of all AGVs within the context "logctx42".
mcClient.trackAgvs((agvId, connectionState, timestamp) => console.log("AGV %o changed connection state to %s at %d", agvId, connectionState, timestamp));

// Stop observing Visualization and State objects.
mcClient.unsubscribe(visSubscriptionId);
mcClient.unsubscribe(stateSubscriptionId);

// Stop client interaction gracefully; disconnect from MQTT broker.
await mcClient.stop();
```


SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library
Usage example – Pub-Sub Abstraction Layer – AGV Client

```
// The target AGV.
const agvId001: AgvId = { manufacturer: "RobotCompany", serialNumber: "001" };

// Create instance of AGV Client "001" with minimal options: communication namespace and broker endpoint address.
const agvClient = new AgvClient(agvId001, { interfaceName: "logctx42", transport: { brokerUrl: "mqtt://mybroker.com:1883" }, vdaVersion: "2.0.0" });

// Start client interaction, connect to MQTT broker.
await agvClient.start();

// Observe Order objects emitted by the Master Control Client.
await agvClient.subscribe(Topic.Order, order => {
    console.log("Order object received: %o", order);

    // Start order handling according to VDA 5050 specification and
    // report order state changes by publishing State objects.
    agvClient.publish(Topic.State, currentState);
});

// Periodically publish Visualization messages with AgvPosition and Velocity.
setInterval(
    () => agvClient.publish(Topic.Visualization,
        { agvPosition: currentPosition, velocity: currentVelocity },
        { dropIfOffline: true }),
    1000);
```


SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library

Usage example – Custom VDA 5050 extension topics and object types

```typescript
// Create instance of Master Controller with minimal client options: communication namespace and broker endpoint address.
const masterController = new MasterController({ interfaceName: "logctx42", transport: { brokerUrl: "mqtt://mybroker.com:1883" }, vdaVersion: "2.0.0" }, {});

// Define extension object type including header properties.
interface MyExtensionObject extends Header {
    key1: number;
    key2: string;
}

// Define a validator function for the extension topic/object (optional).
const myExtensionValidator: ExtensionValidator = (topic, object) => {
    if (typeof object?.key1 !== "number" || typeof object?.key2 !== "string") {
        throw new TypeError("Extension object is not valid");
    }
};

// Register extension topic with validator for both inbound (subscribe) and outbound (publish) communication.
masterController.registerExtensionTopic("myExtension", true, true, myExtensionValidator);

// Start client interaction, connect to MQTT broker.
await masterController.start();

// Observe myExtension messages from all AGVs manufactured by "RobotCompany".
await masterController.subscribe("myExtension",
    { manufacturer: "RobotCompany" },
    (object: ExtensionObject, subject: AgvId, topic: string) => console.log("Extension topic %s with object %o received from AGV %o", topic, object, subject));

// Publish myExtension object to AGV "001".
await masterController.publish("myExtension",
    { manufacturer: "RobotCompany", serialNumber: "001" },
    { key1: 42, key2: "foo" });
```


SIEMENS

Universal VDA 5050 TypeScript/JavaScript Library
Summary & Highlights

Hide complexity of VDA 5050…

- Encapsulate complex control logic of order, action, and state management/handling into reusable components
- Adapt to vehicle-specific navigation & control interfaces through uniform adapters realized by integrators or AGV manufacturers
- Combined or separate use of library components on coordination plane and/or vehicle plane
- Support custom VDA 5050 actions and extension topics/object models

…wherever JavaScript runs

- Cross platform deployments (Linux, Win, macOS, Android, iOS) and runtime environments (IPC, Edge, Cloud, Docker, Industrial Edge, Browser)
- Ease of programming and reuse by *configurable*, *extensible*, and *pluggable* components
- Modern and clean programming of asynchronous operations with async-await pattern
- Complete test coverage of library with unit, component, and integration tests
- Complete API documentation of library
- Available as Open Source on GitHub*

** Library for VDA 5050 released as npm package `vda-5050-lib`*

SIEMENS

Contact

Published by Siemens AG

Dr. Hubertus Hohl
T RDA IOT SES-DE
Otto-Hahn-Ring 6
81739 Munich
Germany

E-mail hubertus.hohl@siemens.com

SIEMENS