Navithor Fleet Control - HTTP API

> *Navitec Systems*

This document contains information on how to configure and use the Navithor HTTP Rest API.

Updated 24.10.2023

Table of contents

Version History

Version	Date	Author	Change
1.0	07.12.2021	Tero Laakso	First version of the API
1.1	04.02.2022	Tero Laakso-Pitkäkoski	Navithor Server.exe configuration moved to ServerSettings.json
1.2	14.02.2022	Tero Laakso-Pitkäkoski	Header levels changed, document main header changed
1.3	23.05.2022	Tero Laakso-Pitkäkoski	CORS parameter description added
1.4	14.10.2022	Aleksi Ålander	Extended the API with new messages
1.5	23.11.2022	Elena Sgonova	Updated license information
1.6	11.05.2023	Toni Liski	Add supported Mission API messages
1.7	05.07.2023	Aleksi Ålander	Added possibility to define pickup shelf on AddProductionOrder and removed not supported message CreateResource
1.8	10.07.2023	Elena Sgonova	Removed not supported messages
1.9	11.07.2023	Aleksi Ålander	Added new messages api/GetRequestedDoorStatus and api/GetAllDoorStatuses
2.0	12.07.2023	Aleksi Ålander	Added new messages api/ClearLocation and api/ClearMachine and added routing for api/AddLoadToLocation and api/AddLoadToMachine to the old resource messages

Version	Date	Author	Change
2.1	19.07.2023	Elena Sgonova	Added mention of GetMissions message
2.2	31.07.2023	Aleksi Ålander	Renamed AddLoadToLocation to LoadAtLocation and AddLoadToMachine to LoadAtMachine and added HTTP GET routing for them
2.3	05.09.2023	Elena Sgonova	Added GetAgvStatus message
2.4	02.10.2023	Toni Liski	Added new REST API messages under api/locations/ and api/machines/ to read loads at the location and the machine.
2.5	24.10.2023	Aleksi Ålander	Added new routing for api/missions DELETE request to abort all missions.

1. General API information

HTTP API is a software interface designed to provide an easy way to control and to gain data from the fleet control system. It serves as an alternative for Navithor MES interface, with a purpose to ease the integration to high level production systems and custom user interfaces. Communication is using HTTP protocol and messages are in JSON format, making them easy to compose and parse with almost any programming language or by human. Basic understanding of these technologies is expected in this documentation.

2. Server configuration

The API has to be enabled before it can be accessed. Using the API does not require a special license.

To enable API, change the **UseWebServer** option to **true** in **ConfigFiles/ServerSettings.json**. **UseHttps** can be set to **true** if secure communication is desired. Secure communication requires certificates to be installed. HTTP API uses the same communication certificates as the Web Client, setting up those certificates is described in **Setting up Web server to enable Web client** document. Although in production environment the secure communication in encouraged, integration to the API is usually easier to start without secure communication enabled. Server has to be restarted after the settings have been changed. The API will start responding soon after the server is up.

2.1. Cross-Origin Resource Sharing (CORS)

When implementing a custom web application that uses HTTP API, server has to allow the application to communicate with it even when the application is not hosted by the server. This mechanism is called CORS and it can be enabled through **ConfigFiles/ServerSettings.json** by setting **AllowCors** option to **true**.

3. HTTP verbs and headers

When making requests to the API four different information types has to be defined by the requesting client. Client has to define the HTTP verb used, this API only uses two of those: **GET** for only receiving info from the server, and **POST** for making the requests to change the state of the system. API path specification section describes a verb for each message.

URL path is the second information that has to be filled by the client application. URL format is the same that is used in web browsing, only difference is the port that has to be manually defined. When requesting version information of the server configured with default settings, and running on same host as the requesting client, URL path would look like this: 'http://localhost:1234/api/GetVersion'.

Headers are a way to provide parameters as a part of the HTTP request. As JSON is the format supported by Navithor in the HTTP body to both directions, **Content-type: application/json** header is expected to be given in every request. The other header expected depends on the authentication method used, those methods are described in following sections.

HTTP body is the field used to provide path specific request parameters for the server. The same body also contains response message created by the server. In all cases body follows the JSON format, and all the information expected in the requests and to be expected in the responses is documented in **API path specification** section.

4. Authentication

Using the API paths requires always authentication. There are two methods for authentication:

- Username and password authentication (Basic)
- API Key authentication

API key authentication is the recommended way to ensure the client application is authorized as it is the most simple approach.

4.1. API key

API key authentication is based on fixed HTTP header provided when making requests to the API. Multiple keys can be configured to the system. Keys are configured in the file located in **ConfigFiles/ServerSettings.json** found in Fleet control installation folder. File with default values is created on first server execution. **RestApiKeys** key is used to define an array of key strings.

HTTP requests should contain the header **ApiKey** with value matching with one of the values defined in **ServerSettings.json**.

4.2. Authentication with username and password

This authentication method makes it possible to authenticate using the same user accounts that are used to login to Fleet control clients (Navithor Client and Web Client). For that purpose there is a specific '/api/token' path. To that path client should POST a form with following text parameters: **username**, **password** and **grant_type**. grant_type should always have value **password**, and the user information has to match with server login credentials. **Content-Type** header also should have value **application/x-www-form-urlencoded**. The following screen capture will show the example with default user credentials.

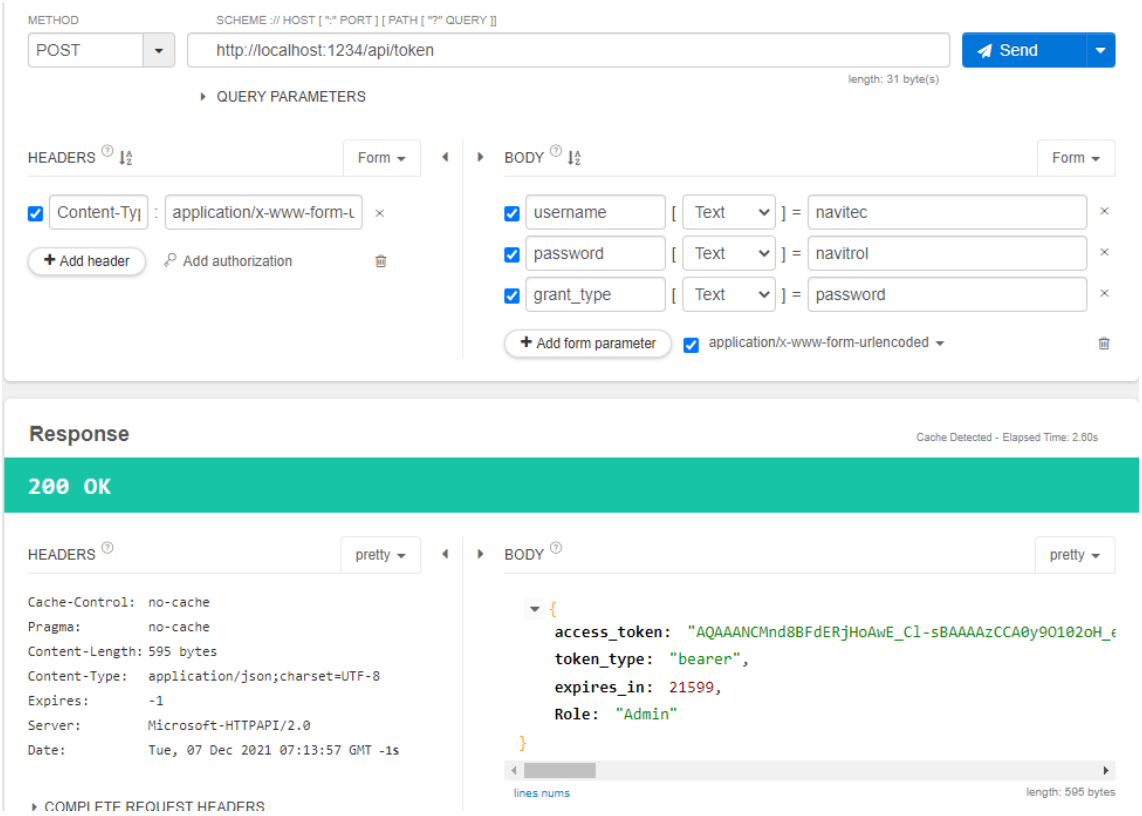

Figure: Toke successfully requested

The string token received with **access_token** should be then set as value for **Authorization** header in the requests to the API. Note that the token has an expiration time after which a new token has to be requested.

5. API specification for general messages

5.1. Fleet control software version

Get version information of server.

URL : `/api/GetVersion`

Method : `GET`

5.1.1. Success Response

Code : `200 OK`

Response example

```
{
    "Major": 2,
    "Minor": 23,
    "Build": 0,
    "Revision": 0
}
```

5.2. Request server to save logs

Request server to save logs. Request can optionally include a description that is included with the log package and machine names. If machines are defined, server tries to get the AGV specific logs from them.

URL : `/api/DumpLogs`

Method : `POST`

Request body parameters

```
{
      "machineNames": "array of strings: Names of the machines - optional",
      "description": "string: Reason why the logs were requested to be saved - optional"
}
```

Request body example

```
{
    "machineNames": ["AGV 1", "AGV 2"],
    "description": "AGVs are having problems with XX"
```

5.2.1. Success Response

Code : `200 OK`

Response body example

Value of **success** indicates if request to save logs was successfully received.

```
{
    "success": true
}
```

5.3. Get area data

Get area data of given machine type.

URL : `/api/GetAreas`

Method : `GET`

Request body parameters

```
{
    "machineType": "string: Machine type name"
}
```

Request body example

```
{
    "machineType": "Omni_demo"
}
```

5.3.1. Success Response

Code : `200 OK`

Response body example

```
{
    "AreaName": "string: Name of the area",
    "AreaVersion": "number: Version number of the area",
    "Level": "number: Level id of the area",
     "Coordinates": "list of points: Coordinates of the area's outside edges",
        "x": "decimal: X-coordinate",
        "y": "decimal: Y-coordinate",
      "Zones": "list of zones: Zones configured in the area",
       "Name": "string: Name of the zone",
          "Coordinates": "list of points: Points of the edges of the polygon",
            "x": "decimal: X-coordinate",
            "y": "decimal: Y-coordinate",
        "ZoneID": "number: ID of the zone",
         "Configurations": "list of decimals: Values for zone configuration"
}
```

5.4. Get environment points

Get environment points of given machine type.

URL : `/api/GetEnvironmentPoints`

Method : `GET`

Request body parameters

```
{
    "machineType": "string: Machine type name",
}
```

Request body example

```
{
    "machineType": "Omni_demo"
```

```
}
```

5.4.1. Success Response

Code : `200 OK`

Response body example

```
{
    "Coordinates": "list of points: Represents the static environment points of the area",
          "x": "decimal: X-coordinate of the envrionment point",
          "y": "decimal: Y-coordinate of the envrionment point",
     "MinX": "decimal: Minimum X-value of all environment points",
     "MaxX": "decimal: Maximum X-value of all environment points",
     "MinY": "decimal: Minimum Y-value of all environment points",
     "MinY": "decimal: Maximum Y-value of all environment points",
}
```

5.5. Get zones of the area

Get all zones congfigured for the area.

URL : `/api/GetEnvironmentZones`

Method : `GET`

5.5.1. Success Response

Code : `200 OK`

Response body example

```
{
    "ZoneResults": "list of ZoneResults: Zones configured
```

```
in the area",
       "Name": "string: Name of the zone",
          "Coordinates": "list of points: Points of the
edges of the polygon",
            "x": "float: X-coordinate",
            "y": "float: Y-coordinate",
        "ZoneType": "number: Type of the zone",
        "ZoneID": "number: ID of the zone",
          "Configurations": "list of doubles: Values for
zone configuration"
}
```

5.6. Request active alarms

Request all active system and machine alarms, or use filters to get only specific alarms.

URL : `/api/errors/{entityId}/{level}/{type}`

Method : `GET`

Possible query parameters are:

entityId (default 0): Any positive integer matching the actual ID of the entity.

level (default NoError):

- NoError
- Warning
- Error
- FatalError

type (default NoGroup):

- NoGroup
- SystemAlarm
- EntityAlarm (*contains machine and symbolicpoint alarms*)
- MachineAlarm
- SymbolicPointAlarm

Examples:

Request all active alarms:

- `/api/errors`

Request all active system alarms:

- `/api/errors?type=SystemAlarm`
- `/api/errors/0/NoError/SystemAlarm`

Request all active alarms of the symbolicpoint ID 100, higher or equal than level *Warning*:

- `/api/errors?type=SymbolicPointAlarm&level=Warning&entityId=100`
- `/api/errors/100/Warning/SymbolicPointAlarm`

Request all active fatal errors of machine ID 1:

- `/api/errors?type=MachineAlarm&level=FatalError&entityId=1`
- `/api/errors/1/FatalError/MachineAlarm`

Note: Using the query parameters any argument can be skipped. With a path format it is not possible to omit arguments in the middle, instead possible query formats are:

- `/api/errors`
- `/api/errors/0/FatalError`
- `/api/errors/100/NoError/SymbolicPointAlarm`

5.6.1. Success Response

Code : `200 OK`

Response body

```
[
    {
        "Id": "number: Alarm Id.",
        "Name": "string: Alarm name.",
        "Description": "string: Alarm description.",
            "ErrorType": "enum: Type of the alarm. See possible type values above.",
         "EntityId": "number: Entity identifier. 0 means system alarm.",
```

```
                "Source": "enum: Unknown, FleetControl,
NavigationSoftware, Supervisor, MachineDevice.",
        "Level": "enum: Level of the alarm. See possible
level values above.",
            "Value": "number: Alarm value (for example
supervisor error ID given by the PLC).",
        "Priority": "number: Alarm priority."
    }
]
```

Response body example

Example alarms:

```
[
    {
        "Id": 3015,
        "Name": "MESInvalidMessageReceived",
          "Description": "Invalid MES message received.
Check logs or entity alarms for more detailed
information.",
        "ErrorType": "SystemAlarm",
        "EntityId": 0,
        "Source": "FleetControl",
        "Level": "Error",
        "Value": 0,
        "Priority": 4
    },
    {
        "Id": 100015,
        "Name": "MachineMissingRequiredCapability",
         "Description": "Machine software version is too
old to support compressed production areas!",
        "ErrorType": "MachineAlarm",
        "EntityId": 1,
        "Source": "FleetControl",
```

```
        "Level": "Error",
        "Value": 0,
        "Priority": 4
    }
]
```

5.6.2. Failure Response

Code : `200 OK`

Response body example

If alarms are not found matching with the given filters, the empty list is returned as a response.

```
[]
```

6. API specification for AGVs

6.1. AGV position initialization

Initialize the position of the AGV to given location.

URL : `/api/InitializePosition`

Method : `POST`

Request body parameters

```
{
    "machineId": "number: identification number integer",
    "x": "number: X coordinate",
    "y": "number: Y coordinate",
    "heading": "number: AGV heading in radians"
}
```

Request body example

```
{
    "machineId": 2,
    "x": 4,
    "y": 12.2,
    "heading": 3.13}
```

6.1.1. Success Response

Code : `200 OK`

Response body example

Value of **success** indicates if position initialization message was successfully delivered to AGV. It does not guarantee that position was accepted as a valid position.

```
{
    "success": true
}
```

6.2. Change AGV production status

Take machines into production, or out of production.

URL : `/api/TakeMachineIntoProduction` - To take machine into production

URL : `/api/TakeMachineOutOfProduction` - To take machine out of production

Method : `POST`

Request body parameters

```
{
       "machineIds": "array of numbers: identification number integers. null value will control the production status of the complete enabled fleet",
}
```

Request body example

```
{
    "machineIds": [5],
}
```

6.2.1. Success Response

Code : `200 OK`

Response body example

Value of **success** indicates if AGVs with given ID(s) were found and could be taken into production, or released from production.

```
{
    "success": true
}
```

6.3. Enable and disable AGV

Change the AGV state to enabled or to disabled.

URL : `/api/EnableMachine` - To enable

URL : `/api/DisableMachine` - To disable

Method : `POST`

Request body parameters

```
{
    "machineId": "number: identification number integer",
}
```

Request body example

```
{
    "machineId": 5,
}
```

6.3.1. Success Response

Code : `200 OK`

Response body example

Value of **success** indicates if AGV with the given ID was found and the vehicle state was changed to enabled or disabled.

```
{
    "success": true
}
```

6.3.2. Failure Response

Code : `400 BadRequest`

Response body example

Machine with given ID was not found.

```
{
    "success": false
}
```

6.4. Release AGV from hold at location

Releases machine currently in hold state at given symbolic point to the given target symbolic point.

URL : `/api/ReleaseHold`

Method : `POST`

Request body parameters

```
{
     "fromId": "number: Identification of symbolic point AGV is released from",
        "targetId": "number: Identification of target symbolic point"
}
```

Request body example

```
{
    "fromId": 1008,
    "targetId": 1009
}
```

6.4.1. Success Response

Code : `200 OK`

Response body example

Currently with all the parameters defined the response is always **true**.

```
{
    "success": true
}
```

6.4.2. Failure Response

Code : `400 BadRequest`

Response body example

Parameters missing.

```
{
    "success": false
}
```

6.5. Release AGV from stop-and-hold location

Releases given AGV from stop-and-hold state (if Navitrol allows this!). Note that stop-and-hold symbolic point has different behaviour than a normal symbolic point, that has hold-rule enabled. Refer to other documentation to understand the different use cases.

URL : `/api/ReleaseStopAndHold`

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
}
```

Request body example

```
{
    "machineName": "AGV 1",
}
```

6.5.1. Success Response

Code : `200 OK`

Response body example

success value indicates if request for the release stop-and-hold was received.

```
{
    "success": true
}
```

6.6. Add a new load to AGV

Add a load to be carried by given AGV. Setting **amount** value to zero will clear the load status of the AGV.

URL : `/api/LoadAtMachine` or `/api/ResourceAtMachine`

Method : `POST`

Request body parameters

```
{
    "machineId": "number: AGV ID",
    "resourceType": "number: Resource type ID",
    "amount": "number: Number of created resources"
}
```

Request body example

```
{
    "machineId": 3,
    "resourceType": 2,
    "amount": 2
}
```

6.6.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if resource was created.

```
{
    "success": true
}
```

6.7. Request load status of a machine

Load status of the AGV is sent as a response, when LoadAtMachine is sent as a GET method. See below also another api call `/api/machines/{id}/loads`.

URL : `/api/LoadAtMachine`

Method : `GET`

Request body parameters

```
{
    "machineId": "number: Id of the AGV (optional, if the name is given)",
    "machineName": "string: Name of the AGV (optional, if the id is given)"
}
```

Request body example

Request load status of the AGV with Id 1:

```
{
    "machineId": 1
}
```

6.7.1. Success Response

Code : `200 OK`

Response body

```
{
    "MachineId": "number: Id of the machine",
    "MachineName": "string: Name of the machine",
     "LoadCount": "number: Total amount of loads on the machine",
     "Load": "LoadData[]: Array of loads on the requested machine. See the data structure below. Default: null"
}
```

LoadData

The data structure for each load returned in the response.

```
{
    "TypeId": "number: Id of the load type",
     "Quantity": "number: Quantity of the load. Default: 1",
     "ShelfId": "number: Shelf Id. Default: 0 if shelves are not defined. If the load is stacked on top of another one, the Shelf Id is increased. The example below has some stacked loads",
     "Side": "Orientation: Side of the rack for 2-sided racks. Default: None",
     "Barcode": "string: Custom identifier for the load. Default: Empty string",
    "LoadId": "number: Unique load Id given by the server for each load when it is created",
     "Slot": "number: Position of the load for example in a buffer lane. Default: 0",
}
```

Response body example

In the example below forklift 1 has one load onboard:

```
{
    "MachineId": 1,
    "MachineName": "forklift 1",
    "LoadCount": 1,
    "Loads": [
        {
            "TypeId": 2,
            "Quantity": 1,
            "ShelfId": 0,
            "Side": "None",
            "Barcode": "",
```

```
            "LoadId": 5349,
            "Slot": 0
        }
    ]
}
```

6.7.2. Failure Response

Code : `200 OK`

Response body example

If the machine Id or name does not match with any machine configured in the system, all fields on the response will be on default values:

```
{
    "MachineId": -1,
    "MachineName": "",
    "LoadCount": 0,
    "Loads": null
}
```

6.8. Request load status of a machine (REST API type)

URL : `/api/machines/{machineId}/loads`

Method : `GET`

To request loads at the machine ID 2:

- `/api/machines/2/loads`

See response examples from above at Request load status of a machine.

6.9. Clear AGV from any loads

Clear all loads that the AGV is currently carrying.

URL : `/api/ClearMachine`

Method : `POST`

Request body parameters

```
{
    "machineId": "number: Id of the AGV (optional, if the name is given)",
    "machineName": "string: Name of the AGV (optional, if the id is given)"
}
```

Request body example

Clear all loads from machine with ID 3:

```
{
    "machineId": 3
}
```

6.9.1. Success Response

Code : `200 OK`

Response body example

```
{
    "success": true
}
```

6.9.2. Failure Response

The response will be a failure, if the given ID does not match with any AGVs configured to the system.

Code : `400 BadRequest`

Response body example

```
{
    "success": false
}
```

6.10. Drive AGV to a symbolic point

Creates an order to drive AGV to the given symbolic point.

URL : `/api/DriveToSymbol`

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
    "machineId": "number: AGV ID",
    "symbolName": "string: Name of the symbolic point"
}
```

Request body example

```
{
    "machineName": "AGV 1",
    "machineId": 1,
    "symbolName": "Pickup Location 1"
}
```

6.10.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if request for the drive order was received.

```
{
    "success": true
}
```

6.11. Drive AGV to XY-location

Creates an order to drive AGV to the given coordinates. The given target needs to be close to a route for it to be accepted.

URL : `/api/SendDriveOrder`

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
      "endPointX": "decimal: X-coordinate of the drive target",
       "endPointY": "string: Y-coordinate of the drive target"
}
```

Request body example

```
{
    "machineName": "AGV 1",
    "endPointX": 124.503,
    "endPointY": -50.234"
}
```

6.11.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if request for the drive order was received.

```
{
    "success": true
}
```

6.12. Request AGV to pickup or dropoff a load

Requests AGV to pickup (LoadMachine) or dropoff (UnloadMachine) a load. AGV has to be already at the correct location before the request.

URL : `/api/LoadMachine` - to pickup load **URL** : `/api/UnloadMachine` - to dropoff load

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
}
```

Request body example

```
{
    "machineName": "AGV 1",
}
```

6.12.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if request for pickup/dropoff command was received.

```
{
    "success": true
}
```

6.13. Request AGV to abort current action

Requests AGV to abort drive order, pickup or dropoff action. In case AGV is not in correct state, request does nothing. AbortDrive tries to abort all current tasks and takes AGV out of production.

URL : `/api/AbortDrive` - to abort drive order (AGV is also taken out of production) **URL** : `/api/AbortLoadMachine` - to abort pickup action **URL** : `/api/AbortUnloadMachine` - to abort dropoff action

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
}
```

Request body example

```
{
    "machineName": "AGV 1",
}
```

6.13.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if abort command was received.

```
{
    "success": true
}
```

6.14. Request AGV to FSTOP or release FSTOP

Requests AGV to do FSTOP triggered by a software. AGV can be released only by another release FSTOP request.

URL : `/api/FSTOPMachine` - to FTSOP the AGV **URL** : `/api/ReleaseFSTOPMachine` - to release AGV from FSTOP

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
}
```

Request body example

```
{
    "machineName": "AGV 1",
}
```

6.14.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if command was received.

```
{
    "success": true
}
```

6.15. Request all AGVs to FSTOP

Requests all AGVs to do FSTOP triggered by a software. AGV can be released only by giving each of them a separate command to be released from FSTOP.

URL : `/api/FSTOPAllMachines`

Method : `POST`

6.15.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if request to FSTOP all AGVs was received.

```
{
    "success": true
}
```

6.16. Request AGV to return to route

Requests AGV to return to route. AGV controller / navigation software is responsible for choosing the optimal trajectory back to the route. Fleet control just gives the request to the controller.

URL : `/api/ReturnToRoute`

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
}
```

Request body example

```
{
    "machineName": "AGV 1",
}
```

6.16.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if request to return AGV to route was received.

```
{
    "success": true
}
```

6.17. Request AGV to go to charge

Requests AGV to go to charge. Fleet control is responsible for checking if this is possible and choosing the charging location.

URL : `/api/RequestChargingForMachine`

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
}
```

Request body example

```
{
    "machineName": "AGV 1",
```

```
}
```

6.17.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if request to take AGV to charge was received.

```
{
    "success": true
}
```

6.18. Request AGV to go to sleep mode

Requests AGV to go to sleep mode. AGV needs to support this feature, fleet control will give the request to the AGV controller. Can be used also to give the request to the whole fleet of AGVs.

URL : `/api/SleepMode`

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name - optional",
    "sleepMode": "boolean: Whether to activate sleep mode (true) or to wake up the AGV from sleep (false)",
     "allMachines": "boolean: Whether to give the request to a single or all AGVs - optional"
}
```

Request body example

```
{
    "machineName": "AGV 1",
    "sleepMode": true
}
```

6.18.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if request to take AGV to charge was received.

```
{
    "success": true
}
```

6.18.2. Failure Response

Code : `400 BadRequest`

Response body example

Request body had neither machine name defined or boolean commandAllMachines as true.

```
{
    "success": false
}
```

7. API specification for production management

NOTE: Starting from Navithor version 3.0.0, order-related API is replaced with Mission API. See more about Mission API further down in the document in the **Mission related messages** section.

Following API routes are removed:

- `/api/AddTransferRequest`
- `/api/EditTransferRequest`
- `/api/DeleteUnhandledOrder`
- `/api/DeleteProductionOrder`
- `/api/DeleteTransferRequest`
- `/api/PauseProductionOrder`
- `/api/PostponeOrder`
- `/api/ContinueProductionOrder`
- `/api/GetProductionOrders`

`AddProductionOrder` request is still supported and input will be converted to the compatible with Mission API format.

7.1. Create a mission with AddProductionOrder message

Requests to create a mission with the given parameters. When AGV ID is not defined, the system will select the suitable AGV for the task. With resource defined, it is moved from pickup locations to target. By just defining target location for the AGV it is possible to order machine to the destination without a resource.

URL : `/api/AddProductionOrder` - To create new mission using old Production order data format.

Method : `POST`

Request body parameters

```
{
    "machineId": "number: AGV identification - optional",
     "targetLocationId": "number: Target symbolic point identification",
    "pickupLocationId": "number: Pickup location symbolic point identification - optional",
```

```
      "resourceTypeId": "number: Type identification of resource - optional",
      "resourceCount": "number: Number of resources - optional",
     "targetShelfId": "number: Target shelf level, -1 if the target is not a shelf - optional",
    "targetShelfOrientation": "string: Orientation at the target rack, if using a sideloader. Options: None, Left, Right - optional",
     "pickupShelfId": "number: Pickup shelf level, -1 if the target is not a shelf - optional",
    "pickupShelfOrientation": "string: Orientation at the pickup rack, if using a sideloader. Options: None, Left, Right - optional",
     "priority": "number: Priority of the order (min 1, max 8) - optional"
}
```

Request body example

```
{
    "machineId": 1,
    "targetLocationId": 1009,
    "pickupLocationId": 1008,
    "resourceTypeId": 1,
    "resourceCount": 1
}
```

7.1.1. Success Response

Code : `200 OK`

Response body example

Value of **result** can have following numberical values:

```
SUCCESS = 0,
INVALID_PICKUP_LOC = 1,
```

```
INVALID_TARGET_LOC = 2,
DATABASE_FAILURE = 3,
RESOURCE_NOT_FOUND = 4,
GENERIC_FAILURE = 5,
```

createdId is a unique identification number for created order. Identification can be used to delete orders.

```
{
    "result": 0,
    "createdId": 9
}
```

7.2. Abort missions

Requests server to abort all missions.

URL : `/api/missions`

Method : `DELETE`

7.2.1. Success Response

Code : `204 No Content`

Request is always accepted and handled on the Server side, so no further information is given back on the reply.

7.3. Mission related messages

Mission API can be used also through the Navithor HTTP API. See the messages and detailed descriptions from the **Navithor 3.0 Mission API** document.

Possible message routes:

- `/api/GetMissions`
- `/api/MissionCreate`
- `/api/MissionStatusRequest`
- `/api/MissionExtend`
- `/api/MissionAbort`

7.4. Request pickup of the resource at location

Fleet control is able to generate production orders based on resource pickup requests.

URL : `/api/ResourcePickupRequest` **URL** : `/api/RequestPickupAtSymbolicPoint`

Method : `POST`

Request body parameters

```
{
    "symbolicPointId": "number: Symbolic point ID",
      "shelfId": "number: Rack shelf identification - optional",
      "orientation": "string: Target rack orientation. Options: None, Left, Right - optional"
}
```

Request body example

```
{
    "symbolicPointId": 3007
}
```

7.4.1. Success Response

Code : `200 OK`

Response body example

success value is always set **true** regardless of the resource status of the location.

```
{
    "success": true
}
```

7.5. Activate or deactivate production break

Fleet control either activates or deactivates production break mode based on this message. Refer to 'Production break' documentation to learn more about the feature.

URL : `/api/ProductionBreakMode`

Method : `POST`

Request body parameters

```
{
      "enabled": "boolean: Indicates whether production break mode should be activated",
}
```

Request body example

```
{
    "enabled": true,
}
```

7.5.1. Success Response

Code : `200 OK`

Response body example

Value of **success** indicates if request was successfully received.

```
{
    "success": true
}
```

8. API specification for symbolic points

8.1. Disable or enable symbolic point

URL : `/api/EnableDisableSymbolicPoint`

Method : `POST`

Request body parameters

```
{
    "symbolicPointId": "number: Symbolic point ID",
    "enabled": "boolean: Is enabled or not"
}
```

Request body example

```
{
    "symbolicPointId": 1008,
    "enabled": true
}
```

8.1.1. Success Response

Code : `200 OK`

Response body example

Request will succeed always when given symbolic point identification is a valid numeric value.

```
{
    "success": true
}
```

8.2. Add a new load to a location

Adds a new load to a certain location. Setting **amount** value to zero will clear the location.

URL : `/api/LoadAtLocation` or `/api/ResourceAtLocation`

Method : `POST`

Request body parameters

```
{
    "symbolicPointId": "number: Symbolic point ID",
    "resourceType": "number: Resource type ID",
    "amount": "number: Number of created resources",
      "shelfId": "number: Rack shelf identification - optional",
      "orientation": "string: Target rack orientation. Options: None, Left, Right - optional"
}
```

Request body example

```
{
    "symbolicPointId": 1003,
    "resourceType": 2,
    "amount": 2,
    "shelfId": -1
}
```

8.2.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if resource was created.

```
{
    "success": true
}
```

8.3. Request load status of a location

Load status of the location is sent as a response, when LoadAtLocation is sent as a GET method. See below also another api call `/api/locations/{id}/loads` .

URL : `/api/LoadAtLocation`

Method : `GET`

Request body parameters

```
{
        "symbolicPointId": "number: Symbolic point ID (optional, if the name or the rackId is given)",
     "symbolicPointName": "string: Name of the symbolic point (optional, if the id or the rackId is given)",
    "rackId": "string: Rack id of the target. Can be used for 2-sided racks instead of the symbolic point id and orientation (optional, if the id or the name is given)"
}
```

Request body example

```
{
    "symbolicPointId": 1045
}
```

8.3.1. Success Response

Code : `200 OK`

Response body

```
{
    "TargetId": "number: Symbolic point Id",
     "RackId": "string: Rack Id of the target. Default: Empty string",
     "LoadCount": "number: Total amount of loads at the symbolic point",
     "Load": "LoadData[]: Array of loads at the requested symbolic point. See the data structure below. Default: null"
}
```

LoadData

The data structure for each load returned in the response.

```
{
    "TypeId": "number: Id of the load type",
     "Quantity": "number: Quantity of the load. Default: 1",
     "ShelfId": "number: Shelf Id. Default: 0 if shelves are not defined. If the load is stacked on top of another one, the Shelf Id is increased. The example below has some stacked loads",
     "Side": "Orientation: Side of the rack for 2-sided racks. Default: None",
     "Barcode": "string: Custom identifier for the load. Default: Empty string",
    "LoadId": "number: Unique load Id given by the server for each load when it is created",
    "Slot": "number: Position of the load for example in a buffer lane. Default: 0",
}
```

Response body example

In the example there are in total 5 loads in the location 1045 that is a buffer lane. The first load is in the beginning of the buffer lane (closest to the symbolic point) in slot 0. The second load is next to it and occupies the slot 1. The last three loads all are in slot 2, but they are stacked on top of each other, which can be seen from the shelfId that goes from 0 to 2. The load with shelfId 2 is on top of the stack.

```
{
    "TargetId": 1045,
    "RackId": "",
    "LoadCount": 5,
    "Loads": [
        {
            "TypeId": 1,
            "Quantity": 1,
            "ShelfId": 0,
```

```
            "Side": "None",
            "Barcode": "",
            "LoadId": 3343,
            "Slot": 0
        },
        {
            "TypeId": 1,
            "Quantity": 1,
            "ShelfId": 0,
            "Side": "None",
            "Barcode": "",
            "LoadId": 3344,
            "Slot": 1
        },
        {
            "TypeId": 1,
            "Quantity": 1,
            "ShelfId": 0,
            "Side": "None",
            "Barcode": "",
            "LoadId": 3345,
            "Slot": 2
        },
        {
            "TypeId": 1,
            "Quantity": 1,
            "ShelfId": 1,
            "Side": "None",
            "Barcode": "",
            "LoadId": 5346,
            "Slot": 2
        },
        {
            "TypeId": 2,
            "Quantity": 1,
```

```
            "ShelfId": 2,
            "Side": "None",
            "Barcode": "",
            "LoadId": 5347,
            "Slot": 2
        }
    ]
}
```

8.3.2. Failure Response

Code : `200 OK`

Response body example

If symbolic point is not found using the given arguments (Id, name or rack Id), all fields on the response will be on default values:

```
{
    "TargetId": -1,
    "RackId": "",
    "LoadCount": 0,
    "Loads": null
}
```

8.4. Request load status of a location (REST API type)

URL : `/api/locations/{id}/loads`

Method : `GET`

To request loads at the symbolicpoint ID 2:

- `/api/locations/2/loads`

To request loads at the rackId 'rack123':

- `/api/locations/rack123/loads`

See response examples from above at Request load status of a location.

8.5. Request load status in the rack shelf (REST API type)

URL : `/api/locations/{id}/loads?shelfId={shelfId}&orientation={orientation}`

Method : `GET`

To request load at the symbolicpoint ID 2 shelf 4:

- `/api/locations/2/loads?shelfId=4`
- `/api/locations/2/loads/4`

To request load at the rackId 'rack234' left side at shelf 4:

- `/api/locations/rack234/loads?shelfId=4&orientation=Left`
- `/api/locations/rack234/loads/4/Left`

Default orientation 'None' is used, if not defined.

See response examples from above at Request load status of a location.

8.6. Clear a location or a shelf from loads

Clears the location from any loads. If a shelf is defined, clears only the specific shelf. If the target is a rack and the shelf is not defined in the arguments, all shelves will be cleared.

URL : `/api/ClearLocation`

Method : `POST`

Request body parameters

```
{
      "symbolicPointId": "number: Symbolic point ID - optional (if name or rackId given)",
     "symbolicPointName": "string: Name of the symbolic point - optional (if id or rackId given)",
    "shelfId": "number: Rack shelf id - optional",
       "orientation": "string: Target rack orientation. Options: None, Left, Right - optional",
    "rackId": "string: Rack id of the target. Can be used for 2-sided racks instead of symbolic point id and
```

```
orientation - optional (if id or name given)"
}
```

Request body example

In most simple use case only symbolic point ID must be given. In following example the request is to clear symbolic point with ID 2001 from all loads:

```
{
    "symbolicPointId": 2001,
}
```

In the second example the request is to clear shelf 3 on the right side of the rack from symbolic point 2001:

```
{
    "symbolicPointId": 2001,
    "shelfId": 3,
    "orientation": "Right"
}
```

In the third example the request is to clear shelf 5 from a rack with id "Aisle_231_Rack_4":

```
{
    "rackId": "Aisle_231_Rack_4",
    "shelfId": 5
}
```

All of the examples above would be valid arguments for the same message in correct context. The arguments can vary based on the use case.

8.6.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if the location was cleared.

```
{
    "success": true
}
```

8.6.2. Failure Response

The response will be a failure, if the symbolic point is not found based on the given arguments, or if the shelf is defined, but it is not found for the symbolic point.

Code : `400 BadRequest`

Response body example

```
{
    "success": false
}
```

8.7. Report current door status

Doors are special type of symbolic points, defined using Navithor Tools. To report the current status of the door, this request can be used. Refer to Navithor document "Tag Statuses" for configuring the doors.

URL : `/api/DoorStatus`

Method : `POST`

Request body parameters

Either the name or ID of the symbolic point has to be defined. Name is prioritized when both values are defined.

```
{
      "symbolicPointId": "number: Symbolic point ID - optional",
     "symbolicPointName": "string: Symbolic point name - optional",
    "doorState": "number: Status of the door"
}
```

Door status can have one of these values:

```
offline = 0,
closed = 1,
closing = 2,
opening = 3,
open = 4,
```

Request body example

```
{
    "symbolicPointId": 3007,
    "doorState": 4
}
```

8.7.1. Success Response

Code : `200 OK`

Response body example

success value is always set to **true** when request is responded with code 200.

```
{
    "success": true
}
```

8.7.2. Failure Response

Code : `400 BadRequest`

Response body example

Request body had no symbolic point ID defined or symbolic point with the given ID does not exist in the system.

```
{
    "success": false
}
```

8.8. Get requested door status

Get the currently requested door status. The message is required to control the doors according to the requests from the fleet control. The message returns both requested and current status of the door.

URL : `/api/GetRequestedDoorStatus`

Method : `GET`

Request body parameters

```
{
      "symbolicPointId": "number: Symbolic point ID - optional (if name given)",
     "symbolicPointName": "string: Symbolic point name - optional (if id given)"
}
```

Request body example

```
{
    "symbolicPointId": 5001
}
```

8.8.1. Success Response

Code : `200 OK`

Response body

```
{
    "SymbolicPointId": "number: Symbolic point ID",
    "SymbolicPointName": "string: Symbolic point name",
      "CurrentDoorState": "number: Current state of the door, 1 = closed, 4 = open",
    "RequestedDoorState": "number: Requested state of the door, 1 = door should be closed, 4 = door should be open"
}
```

Response body example

The door 5001 is currently closed, but it is requested to be open:

```
{
    "SymbolicPointId": 5001,
    "SymbolicPointName": "Door 1 (5001)",
    "CurrentDoorState": 1,
    "RequestedDoorState": 4
}
```

8.8.2. Failure Response

Code : `200 OK`

Response body example

If the input in the request is invalid or the symbolic point ID or the name does not match with any doors configured to the system, the response message indicates this with all number values being -1.

In this case the response will always contain following data:

```
{
    "SymbolicPointId": -1,
    "SymbolicPointName": "",
    "CurrentDoorState": -1,
    "RequestedDoorState": -1
}
```

8.9. Get all door statuses

Get the current status and the requested status for all doors configured to the system. The message can be used in combination with SetDoorStatus message to control the doors.

If there are no doors configured, the response will be an empty list.

URL : `/api/GetAllDoorStatuses`

Method : `GET`

8.9.1. Success Response

Code : `200 OK`

Response body

The response contains a list of all door statuses:

```
[
    {
        "SymbolicPointId": "number: Symbolic point ID",
            "SymbolicPointName": "string: Symbolic point name",
        "CurrentDoorState": "number: Current state of the door, 1 = closed, 4 = open",
        "RequestedDoorState": "number: Requested state of the door, 1 = door should be closed, 4 = door should be open"
    }
]
```

Response body example

The first door is closed and is also requested to be closed. The second door is open and is requested to be open. The third door is currently closed, but it is requested to be open.

```
[
    {
        "SymbolicPointId": 5000,
        "SymbolicPointName": "First door (5000)",
        "CurrentDoorState": 1,
        "RequestedDoorState": 1
    },
    {
        "SymbolicPointId": 5003,
        "SymbolicPointName": "Second door (5003)",
        "CurrentDoorState": 4,
        "RequestedDoorState": 4
```

```
    },
    {
        "SymbolicPointId": 5004,
        "SymbolicPointName": "Third door (5004)",
        "CurrentDoorState": 1,
        "RequestedDoorState": 4
    }
]
```

9. API specification for resources

9.1. Modify resource type

Modifies resource type of an existing resource in the system.

URL : `/api/ModifyResourceType`

Method : `POST`

Request body parameters

```
{
    "resourceId": "number: Resource ID",
    "nextResourceTypeId": "number: New resource type ID"
}
```

Request body example

```
{
    "resourceId": 20301,
    "nextResourceTypeId": 25,
}
```

9.1.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if resource type was modified.

```
{
    "success": true
}
```

9.2. Modify resource quantity

Modifies resource quantity of an existing resource in the system.

URL : `/api/ModifyResourceQuantity`

Method : `POST`

Request body parameters

```
{
    "resourceId": "number: Resource ID",
        "nextResourceQuantity":  "number:  New  resource
quantity"
}
```

Request body example

```
{
    "resourceId": 20301,
    "nextResourceQuantity": 3,
}
```

9.2.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if resource quantity was modified.

```
{
    "success": true
}
```

9.3. Delete resource

Removes resource from the system completely.

URL : `/api/UntrackResource`

Method : `POST`

Request body parameters

```
{
    "resourceId": "number: Resource ID",
}
```

Request body example

```
{
    "resourceId": 20301,
}
```

9.3.1. Success Response

Code : `200 OK`

Response body example

success value will indicate if resource was deleted.

```
{
    "success": true
}
```

9.4. Get status of a resource

Request status of a resource from Fleet control.

URL : `/api/GetResourceStatus`

Method : `GET`

Request body parameters

```
{
    "resourceId": "number: Resource ID",
}
```

Request body example

```
{
    "resourceId": 20301
}
```

9.4.1. Success Response

Code : `200 OK`

Response body example

```
{
    "ResourceId": 234,
    "ResourceTypeId": 2,
    "PreviousHolder": null,
    "CurrentHolder": 100,
    "CurrentQuantity": 1,
    "Deleted": 2,
    "ShelfId": -1,
    "ResourceIdentifier": ""
}
```

9.5. Set status of the resource

Set the status of the load at the symbolic point or in the rack. See more details from Navithor 3.0 Mission API document.

URL : `/api/LocationSetLoadStatus`

Method : `POST`

Request body parameters

```
{
    "TargetId": "string: Target symbolic point id. Either this or RackId needs to be set.",
    "RackId": "string: Rack id of the target. Either this or TargetId needs to be set.",
      "Loads": "LoadData[]: Load to set at the given target. See Data Structures section.",
}
```

LoadData

```
{
     "TypeId": "int: Id of the resource type to set. Set to 0 to clear. Required.",
    "Quantity": "int: Quantity of the load. Default: 1",
        "ShelfId": "int: shelf Id, Default: no shelf defined.",
       "Side": "OrientationEnum: side of the rack for sideloaders. Default: None.",
      "Barcode": "string: identifier for the resource. Default: empty string."
}
```

Request body example

```
{
    "TargetId": "12",
    "Loads": [
        {
            "TypeId": 1,
        },
    ],
}
```

9.5.1. Success Response

Code : `200 OK`

Response body example

success value indicates request was received successfully.

```
{
    "success": true
}
```

9.6. Get status of the AGV

Request status of an AGV from Fleet control.

URL : `/api/GetAgvStatus`

Method : `POST`

Request body parameters

```
{
    "machineName": "string: AGV name",
    "machineId": "number: AGV ID",
}
```

Request body example

```
{
    "machineName": "forklift 2",
    "machineId": 2,
}
```

9.6.1. Success Response

Code : `200 OK`

Response body example

```
{
    "Success": true,
    "Data": {
        "Name": "forklift 2",
        "MachineId": 2,
        "PositionInfo": {
            "X": -17.499799728393555,
            "Y": 18.788017272949219,
            "H": 1.5707962830835065,
            "Level": 0
        },
        "TargetInfo": {
```

```
            "IsMachineAtTarget": false,
            "MachineAtLocationId": -1,
            "TargetLocationId": 14,
            "TargetShelfId": 2,
            "DistanceToTarget": 12.92915353178978
        },
        "IsLoadOnboard": true,
        "Loads": [
            {
                "LoadTypeId": 1,
                "LoadId": 119,
                "Barcode": "",
                "Slot": 0
            }
        ],
        "PositionConfidence": 97,
        "BatteryLevel": 100.0,
        "Speed": -1.0,
        "State": "AUTO",
        "CurrentTask": "Drive",
        "InAuto": true,
        "InManual": false,
        "IsOperational": true,
        "InProduction": true,
        "Enabled": true,
        "IsCharging": false,
        "ChargingRequested": false
    }
}
```


9.7. Activate mission template

Message is used to activate existing mission templates from the `MissionTemplates` folder at the Navithor Server root. Templates can be reloaded into server memory using **Reload overrides** functionality.

Note: ExternalId and MachineId values provided in the message will overwrite the values in the mission template, which is loaded into memory.

URL : `/api/ActivateMissionTemplate`

Method : `POST`

Request body parameters

```
{
     "TemplateName": "string: Template filename with or without `.json` extension.",
    "ExternalId": "string: External ID for mission to be created or empty string if value from mission template should be used.",
     "MachineId": "int: Machine ID which should execute the mission or `0` if value from mission template should be used.",
}
```

9.7.1. Success Response

Code : `200 OK`

Response body example

success value indicates request was received successfully.

```
{
    "ExternalId": "string: Unique ID for a mission, given by client.",
    "InternalId": "int: Unique ID for a mission, given by server.",
    "Success": "bool: Result value signals if request was successful.",
     "Description": "string: Information whether result was succesful or reason for failure.",
}
```

10. Getting started with the integration

Because the nature of the HTTP protocol and how the JSON messages are composed, testing the interface is easy without writing any software. There are a plenty of tools that allow you to easily create HTTP requests with a correct body and headers. Just use your web search engine to find one. Using this kind of manual tools is usually a good first step to make sure the server is configured properly, you have understood the specification and the request is performing as you are expecting it to work.