Interreg
North-West Europe
eHUBS
European Regional Development Fund

EUROPEAN UNION

An API standard for the information dashboard for users on an eHUB kiosk & website platform

Deliverable 6.3

December 2020

Tjalle Groen (Taxistop)

Summary sheet

Project Name	**eHUBS**
Title of the document	An API standard for the information dashboard for users on an eHUB kiosk & website platform
Deliverable	Deliverable 6.3
Work Package	Long term
Programme	Interreg North-West Europe
Coordinator	City of Amsterdam
Website	http://www.nweurope.eu/projects/project-search/ehubs-smart-shared-green-mobility-hubs/
Author	Tjalle Groen
Status	Draft
Dissemination level	Public
Reviewed by	Luk Roose; Elnert Coenegrachts
Submission date	December 2020
Starting date	January 2019
Number of months	36

Project partners

Organisation	Abbreviation	Country
Gemeente Amsterdam	AMS	The Netherlands
Promotion of Operation Links with Integrated Services aisbl (POLIS)	POLIS	Europe
Taxistop asbl	Taxi	Belgium
Autodelen.net	Auton	Belgium
Bayern Innovativ GMbH	BI	Germany
Cargoroo	CA	The Netherlands
URBEE (E-bike network Amsterdam BV)	URBEE	The Netherlands
Gemeente Nijmegen	NIJ	The Netherlands
Transport for the Greater Manchester	TfGM	Great Britain
Stad Leuven	LEU	Belgium
TU Delft	TUD	The Netherlands
University of Newcastle upon Tyne	UN	Great Britain
Ville de Dreux	DR	France
Stadt Kempten (Allgäu)	Kemp	Germany
Universiteit Antwerpen	UAntwerp	Belgium

Document history

Version	Date	Organisation	Main area of changes	Comments
0.1	30/10/2020	Taxistop	First draft	
0.2	06/12/2020	Taxistop	Second draft	Update to TOMP v1.1
0.3	15/12/2020	Taxistop	Third draft	Added chapter on the KIOSK
0.4	18/12/2020	Taxistop	Final version	

Table of Contents

1. Introduction

The goal of work package long term effects, deliverable 6.3 is not only to create an API standard for the information dashboard but also to create the first version of a KIOSK application to showcase this API and to inform users about the Mobility options available at each eHUB.

The following document describes the API standard, defined as a subset of the TOMP-API described and defined in work package long term effects, deliverable 6.2.

The TOMP-API is under continuous development, therefore it is recommended to keep up to date with the latest developments of this standard especially when implementing.

This document is based on the Dragonfly 1.1 release from December 2020 and tries to paint a complete picture, but we would like to refer to the official TOMP sources for the latest version.

To do so please follow one of the links below:

The swaggerhub:
https://app.swaggerhub.com/apis-docs/TOMP-API-WG/transport-operator_maas_provider_api/

The wiki – blueprint:
https://github.com/TOMP-WG/TOMP-API/wiki/Introduction

This API will be used to integrate multiple TO's in the information dashboard for users on an eHUB kiosk & website platform. Going forward described as the KIOSK.

2. The API – a part of the TOMP-API

The TOMP-API is an interface to enable communication between Transport Operators and MaaS Providers. Therefore this as an extensive collection of protocols covering the whole MaaS ecosystem from both sides.
Adhering to the TOMP-API will enable near-seamless integration, sparing the burden of single case integration for each separate TO or MP.

Within WP D6.3 we have created an API standard for the information dashboard for users on an eHUB kiosk & website platform. From now on we can describe this as the KIOSK.

The figure below gives a quick overview of the TOMP ecosystem, the API enables complete integration within a MaaS system, with information exchange between both parties for all components.

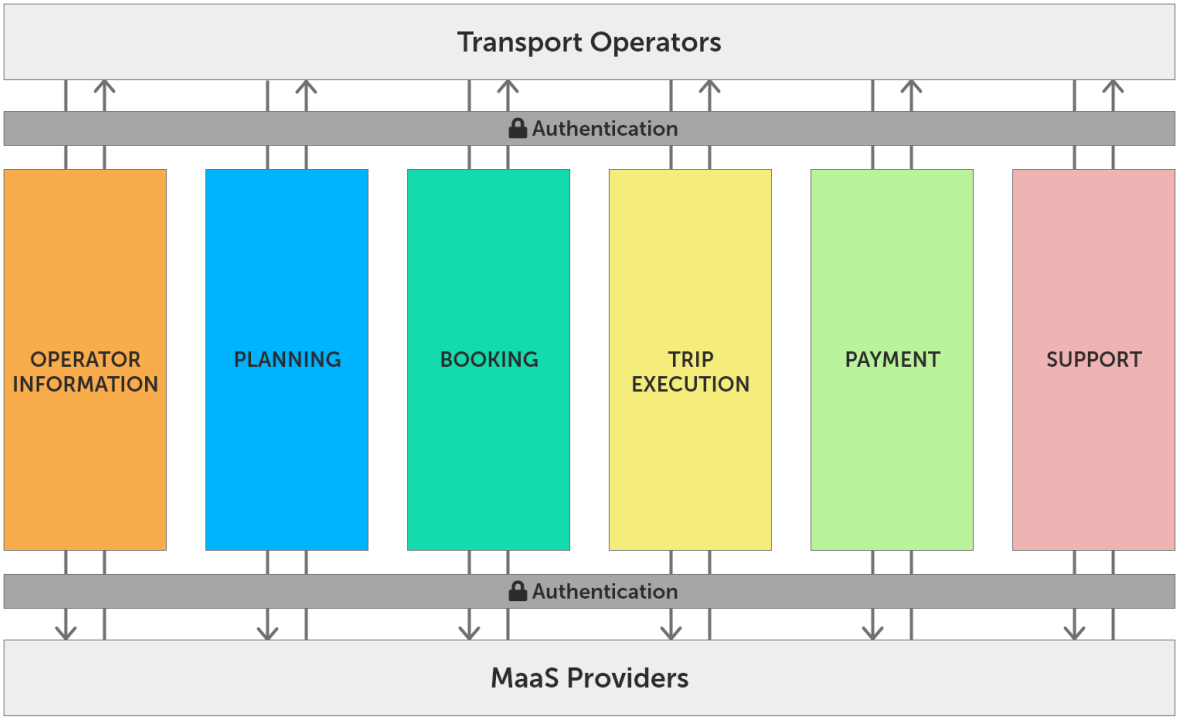

Figure 1: overview of the TOMP-API – Tjalle Groen

If a transport operator wants to get ready for the full integration within MaaS a the full implementation of the TOMP-API is recommended.
Because of the modular setup from the system, it is possible to implement only the parts relevant for your needs.

We consider the model for the levels of MaaS integration by Steve Sarina as depicted below. The TOMP-API aims to fulfil all 4 levels of integration, the current state of development lies between **level 2: Integration of booking** & payment and **level 3: Integration of the service offer**.

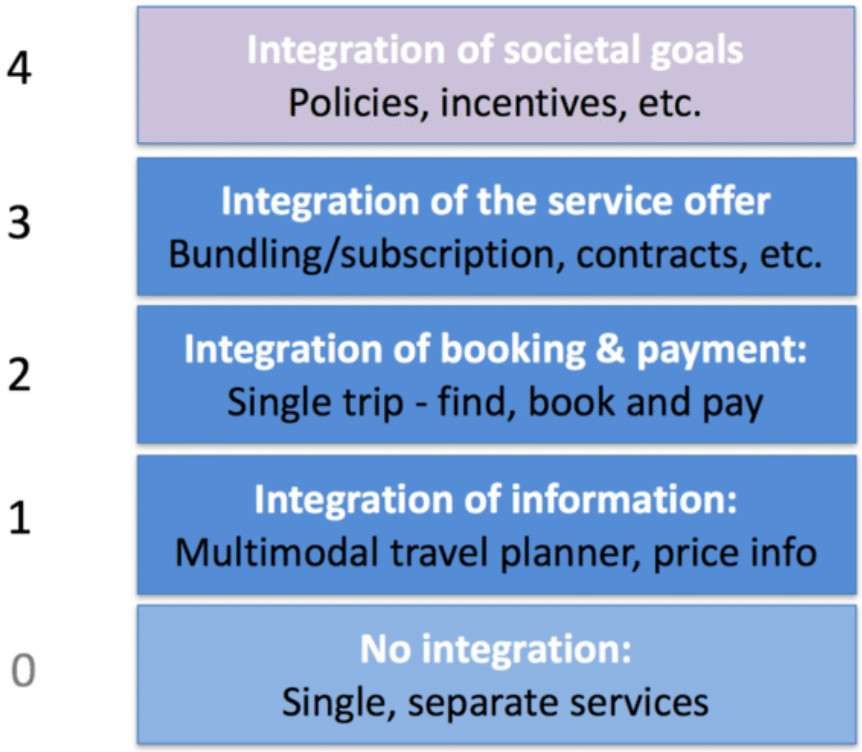

Figure 2: 4 levels of MaaS - source: Steven Sarasini (https://www.researchgate.net/figure/Proposed-topology-of-MaaS-including-Levels-0-4-left-and-examples-right_fig2_320107637).

The KIOSK application focused on providing information to users. There is no booking or route-planning foreseen within the KIOSK.
Therefore the level of integration required is **Level 1: Integration of information**.
This is why we will focus on the building blocks required to make this possible. The definition of what is implemented and how this part can be used is to be described in the operator/meta endpoint.

The required blocks for the eHUBS KIOSK are:

Operation-Information: Static information on the API-owner.

Planning: Up to date (realtime when possible) information on availability, estimated travel time and costs of assets.

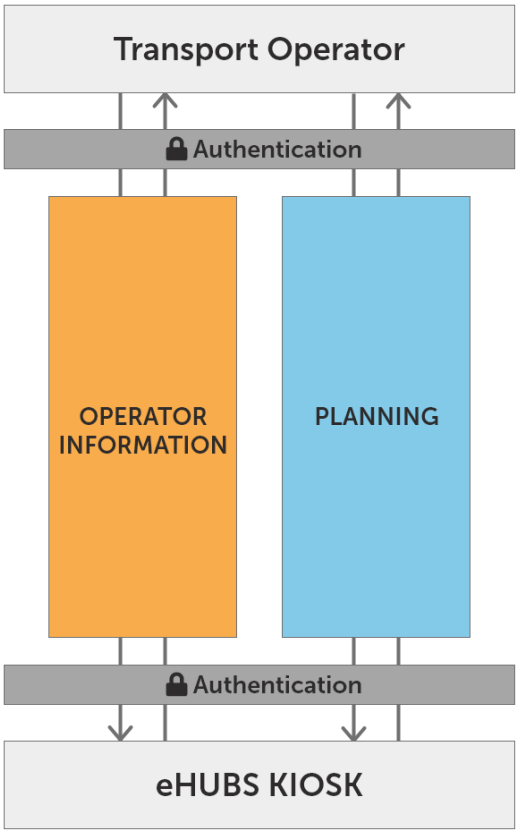

Figure 3: an overview of pillars required for eHUBS integration – Tjalle Groen

3. Transport Operator classifications

There are two main types of Transport Operators we consider in our system, station-based and free-floating. They are both covered in the TOMP-API system but need a slightly different implementation.

a. Plannable journeys

This category revolves around trips that don't require a fixed destination and have either a fixed or a not fixed departure point)

I. Station-Based

A physical eHUB can house a station for multiple Transport Operators, ranging from e-bikes to EV's. These stations will have a standard occupancy of certain vehicles and will most likely be for round trip usage (return to the same point).

In this case, there are different options to implement this, either a full on-demand approach where you use the planning option to look for available assets or a more static approach via the operation information section.

II. Free-floating (= on Demand)

If a TO has different assets spread over a certain geographical area we can call this a free-floating service. Assets do not have to be returned to a specific location, they can be left anywhere within a (usually geofenced) area.
An interested user has to locate the asset on a map and unlock to operate.

b. Predefined Transport

There are multiple mobility options with a fixed destination & departure point.

I. Public transport

A system of vehicles such as buses and trains that operate according to a regular timetable on fixed routes and are to be used by the general public.

II. Carpooling

Also known as ridesharing, a group of people travelling together in a car to reach a joint destination, generally for work or school. But also very effective for leisure activities. Some adjustments to the pre-defined ride are possible, with limitations.

4. Standardisation

The TOMP-API adaptation is a work in progress for the eHUBS pilot cities. Therefore we cannot solely rely on the API described in this document.
This problem has to be tackled by a custom implementation for certain Transport Operators, this, of course, is very time consuming and thus expensive.

The different implementation cases can be roughly divided into two categories:

1. Public Transport & supported standards:

An eHUB cannot function without access to Public Transport data, and the process of letting them provide their data in a TOMP-ready way is tedious and beyond our control.
Therefore there is no other way for a user to get this data directly from them by implementing it how it is provided.
For Belgium these Operators are: NMBS/SNCB, De Lijn, TEC, STIC & MIVB.

Because of existing standards, it is possible to integrate different systems in a fairly seamless way. These standards can be industry-driven (f.e. GTFS & GTFS-RT) or European standards like NETEX and SIRI. The focus lies on getting as compliant as possible with the existing standards to optimize reuse when expanding across borders.
These standards require large file transfers of the full network which need to be imported in the database and require regular updates.

This implies a great effort, with recurring adaptations. To make this feasible an implementation & maintenance cost is required.

2. API's that requires custom integration

Depending on the location and the local policies there might be Operators active in an eHUB area who have not yet implemented the TOMP-API. If they want to be connected to the eHUB there is custom connection code writing to be done. The cost for this is to be calculated outside the common installation/maintenance cost of eHUB itself.
An eHUB will be placed on request of the Municipality, they are thus responsible for the data that needs to be imported.

Depending on the size and willingness to cooperate, the following scenarios are possible.

- One-off integration
- Reusable Integration
- Mapping to the TOMP-API

The table below shows the pro/contra analysis of the different methods.

	Reusability	Workload	Benefit	Desirability
TOMP-API	+++	-	All	++
One-off	-	++	TO	-
Reusable	+	++	TO & eHUB	+
Mapping	++	++	TO & eHUB	+

Table 1: Integration types analysis

One-off integration:

If there is a requirement to include an Operator which is only active / or wishes to be connected into one eHUB or in one eHUB region.
The implementation is completely made-to-measure for the one region

Reusable Integration :

In most cases, we can connect a single operator to multiple eHUB-regions. This means that the bulk of the integration only needs to be done once for multiple locations.

Mapping to the TOMP-API

Another way to handle the integration is to provide Data-Mapping to the TOMP-API.
Data mapping for an API is the process of converting (or translating) source field names into the target field names. This also includes the documentation of any data transformation logic.

The mapping could be shared with the Transport Operator to broaden the implementation of the TOMP-API and reinforcing the operator's strategic advantage in the MaaS ecosystem.

Across the European continent, there are multiple standards in use for communication of either specific kinds of Mobility (ISXI for carsharing, GBFS+ for bike-sharing,…) or more general and extensive standards like NETEX (a requirement for all major mobility operators for all European member states) & SIRI (European choice for Real-Time exchange). There are even local standards (OSLO in Flanders). For these standards, the same methodology of data-mapping could be used.

With one effort per standard, we could include all Transport Operators adhering to this standard in a relatively simple manner.

Conclusion:

When no other options remain, custom integration will prove to be inevitable.
If this is the case however we should strive for maximum reusability. This can be done by adding one operator to multiple eHUBS or by working together with the Transport to create a mapping they can use for further projects. The choice is to be made per Operator or Standard.

5. Implementation requirements

In this chapter, we will take a deeper look into the specifics of the standards and how they are to be implemented.
Warning: this part contains code, it's not scary by necessary for this part.

As determined in chapter three there are a few different approaches to implementation possible, this depends on the type of service ran by a Transport Operator.

What we are displaying here are the minimum implementation guidelines for TOMP implementation so they can be integrated within the digital side of the eHUBS project in the most automated fashion. It is highly recommended though to study the TOMP-API's online documentation for the most up to date version. The implementation is not limited to what is described in this document.

a. Endpoint description

The endpoints needed for implementation are :

Operator Information:

GET /operator/meta

This endpoint describes the meta-information of what about the extent of TOMP-API readiness. A description of all the endpoints that are available.

GET /operator/information

This endpoint describes the Operator itself and the requirements for booking/planning.

GET /operator/stations

An overview of available stations for asset pickup.

GET operator/available-assets

The endpoint providing available assets at a certain station/location. This is a near real-time endpoint, the information needs to be up to date.

Planning

POST /plannings

The planning endpoint forms the exploration phase of a trip. The perfect place to search for real-time availability of specific assets.

The table below overviews which endpoint is required per type of integration. The top side lays out the endpoints, this left side the types of integration.

	operator /meta	operator /information	operator /stations	operator /available-assets	plannings
On-demand	x	x			x
Station based	x	x	x	x	
Combination	x	x	x	x	x

Table 2. Required endpoints per integration type

b. The endpoint breakdown

We will start with a breakdown JSON result of the different endpoints.

GET /operator/meta

Example JSON response:

```
{
        "version": "1.0.0",
        "baseUrl": "https://your-api.org/",
        "endpoints": [{
                        "method": "POST",
                        "path": "/plannings/",
                        "status": "IMPLEMENTED"
                },
                {
                        "method": "GET",
                        "path": "/operator/meta",
                        "status": "IMPLEMENTED"
                },
                {
                        "method": "GET",
                        "path": "/operator/information/",
                        "status": "IMPLEMENTED"
                }
        ],
        "scenarios": [
                "POSTPONED_COMMIT"
```

```
        ],
        "processIdentifiers": {
                "planning": [
                        "PLANNING_BASED"
                ]
        }
}
```


- **version**: the current version of the API you are offering
- **baseURL**: the URL where the endpoints need to be called
- **endpoints**: a list of the available endpoints
    - method: the HTTP method required to call the function
    - path: the path of the webservice call, to be used in combination with the baseURL
    - status: current state of implementation
        - IMPLEMENTED – ready to use
        - NOT_IMPLEMENTED – this endpoint is not available
        - PLANNED – not available yet, but will be in the future.
- **scenarios:**
- **processIdentifiers**:
    - planning: the planning identifier
        - PLANNING_BASED
        - ASSET_BASED

GET operator/information


```
{
        "systemId": "XXTO0001",
        "language": [
                "fr-FR"
        ],
        "name": "FreeBike",
        "shortName": "FB",
        "operator": "FreeBike",
        "url": "https://www.rentmyfreebike.com",
        "purchaseUrl": "https://www.rentmyfreebike.com/purchase",
        "startDate": "2020-11-16",
        "phoneNumber": "555-12345",
        "email": "rent@freebike.com",
        "timezone": "IST",
        "licenseUrl": "https://www.rentmyfreebike.com/license",
        "typeOfSystem": "FREE_FLOATING",
        "chamberOfCommerceInfo": {
                "number": "string",
                "place": "string"
        },
        "conditions": "string",
        "productType": "RENTAL",
        "assetClasses": [
                "AIR"
```

```
        ]
}
```

- **systemID**: Globally unique identifier for the Transport Operator
- **language**: a list of supported languages – these can be requested with the Accept-Language header
- **name:** Name of the organisation (Should match Content-Language)
- **shortName:** The preferred short name for your organisation.
- **operator:** the name of the operator (can be different from the organisation name)
- **URL:** the fully qualified URL of the Transport Operator (must include HTTP/HTTPS)
- **purchaseUrl:** the fully qualified URL for booking/membership
- **startDate**: YYYY-mm-dd – when did you start operating
- **phoneNumber**: the full (including country code) phone number of your operations
- **email**: contact email address
- **timezone**: the timezone you are operating in (EST – European Standard Time)
- **licenseUrl**: Possible licensing information (fully qualified URL)
- **typeOfSystem**
  - FREE_FLOATING
  - STATION_BASED
- **conditions**: possible conditions for booking
- **productType:** what kind of product are you offering
  - RENTAL: one person/group use of the asset
  - SHARING: If the asset is shared with others, f.e. Carpool, public transport, …

POST plannings

Planning is a POST call, indicating information is being transferred when doing the request. This is done via a JSON body.
For the purpose of the eHUBS KIOSK API the query parameter: booking-intent = false is required.
Note that this JSON body does not use all (non required) parameters used in the TOMP-API description, this is because that information is not in use for this operation to run.

```
POST https://exampleTO.com/plannings/?booking-intent=false
{
  "from": { "coordinates": { "lng": 6.169639,"lat": 52.253279} },
  "radius": 100,
  "startTime": "2020-06-24T07:12:03.000Z",
}
```

- **from**: the departure coordinates, in LNG,LAT format. This is the actual location of the eHUB.

- **radius**: the radius around the eHUB that a user is prepared to travel in meters.
- **startTime**: time when the rental starts, default is now.

The response :


```
{
        "validUntil": "2020-12-31T15:38:42.941Z",
        "options": [{
                "id": "string",
                "from": {
                        "name": "string",
                        "stationId": "string",
                        "coordinates": {
                                "lng": 6.169639,
                                "lat": 52.253279
                        },
                        "physicalAddress": {
                                "streetAddress": "example street 18, 2nd floor, 18-B33",
                                "areaReference": "Smallcity, Pinetree county",
                                "postalCode": "string",
                                "country": "NL"
                        }
                },

                "state": "NEW",
                "legs": [
                        {
                                "id": 1234abc
                                "from": {
                                        "lat": "50.86453",
                                        "lng": "4.68077"
                                },
                                "to": {
                                        "lat": "50.99763",
                                        "lng": "4.753283"
                                },
                                "departureTime": "2020-12-06T14:38:42.942Z",
                                "arrivalTime": "2020-12-06T14:38:42.942Z",
                                "assetType": {
                                        "type": "carpool",
                                        "name": "Carpool to: Tremelo",
                                        "assetClass": "OTHER",
                                        "assetSubClass": "Carpool",
                                        "co2-per-km": 110,
                                        "smoking": false,
                                        "persons": 2
                                },

                                "asset": {
                                        "assetId": "672643/8371920",
                                        "name": "Bart",
                                        "image":
"https://www.carpool.be/pics/users/avatars/avatar003.png",
                                        "rentalUrl":
"https://www.carpool.be/rides/ad/672643/8371920?sj=eyJzdGF0dXMiOiJPSyIsInByZWYiOiIyIiwiZnJvbSI
6eyJsYXQiOjUwLjg2MjgxMiwibG9uIjo0LjY5NTQ4OSwiY291bnRyeSI6IkJFIiwiY2l0eSI6IkxldXZlbiIsInBvc3Rhb
CI6IjMwMDEiLCJzdHJlZXQiOiJQcmlucyBEZSBMaWduZXN0cmFhdCJ9LCJ0byI6eyJjb3VudHJ5IjoiQkUiLCJjaXR5Ijo
```

```
iVHJlbWVsbyIsInBvc3RhbCI6IjMxMjgiLCJzdHJlZXQiOiJBYW4gZGUgQmVyZyIsImxhdCI6NTAuOTk3NjMsImxvbiI6N
C43NTMyODN9fQ==",
                                        "place": {
                                                "physical-address": {
                                                        "address": "Aan de Berg",
                                                        "city": "Tremelo",
                                                        "postalCode": "3128",
                                                        "country": "BE",
                                                        "type": "destination"
                                                }
                                        }
                                },
                                "pricing": {
                                        "estimated": true,
                                        "description": "The price may vary when booking,
this is the price for the total ride of the driver.",
                                        "class": "carpool_price",
                                        "parts": [{
                                                "name": "Price traject driver",
                                                "amount": 2.2,
                                                "currencyCode": "EUR",
                                                "type": "FIXED",
                                                "unitType": "KM",
                                                "class": "carpoolPriceDriver"
                                        }]
                                }
                        }
                ]
        }]
}
```


- **validUntil:** How long is this result valid
- **options**: a list of travel options for the request
    - **id**: A unique ID for the presented option
    - **from:** Departure point
    - **state**: The current state of this options
    - **legs**: A list of legs within the result
        - **id**: Unique ID for the journeyleg
        - **from:** Departure point
        - **to:** Arrival point
        - **assetType**:
            - **Id:** Unique ID to identify this result
            - **stationId**: Unique identifier for the station
            - **nrAvailable**: Total number of assets available at the station
            - **assets**: A description of the assets
                - **Id**: A unique identifier for the asset
                - **isReserved**: A boolan to indicate if the asset is reserved or not
                - **isReservedFrom**: If reserved a full date with timezone
                - **isReservedTo**: If reserved a full date with timezone

    - **isDisabled:** Is the asset in use (fe. In case of maintenance)
    - **overriddenProperties:** A list of properties that are different then the ones provided in the sharedProperties field for the station
        - **name:** a name for the asset
        - **location:** a location for the asset
        - **rentalUrl:** a direct URL where the asset can be booked.
        - **co2perKm:** co2 usage per km
        - **persons:** Number of people who can use the asset
        - **image:** an image url
        - **…**
- **assetClass**: The asset class, derived from the NETEX standard.
  `AIR, BUS, TROLLEYBUS, TRAM, COACH, RAIL, INTERCITYRAIL, URBANRAIL, METRO, WATER, CABLEWAY, FUNICULAR, TAXI, SELFDRIVE, FOOT, BICYCLE, MOTORCYCLE, CAR, SHUTTLE, OTHER, PARKING, MOPED, STEP`
- **assetSubClass**: In the case of OTHER as assetClass – specify the class here.
- **sharedProperties**: a list of properties shared among the available assets at the station

Note: these calls & responses only hold the information required for our purposes, check the latest version of swaggerhub for a full overview of possible values.

GET operator/stations

Query parameters:

- **coordinates:** (optional) – a string in the format: "coordinates": { "lng": 6.169639,"lat": 52.253279} of the location of the eHUB.
- **radius:** (optional) – a radius in meters where the stations should be located
- **Offset:** default 0 (paging is supported from v1.1 – the start of result)
- **Limit:** default 20 (paging is supported from v1.1 – the maximum number of results)

Note: coordinates & radius are an extension on the TOMP-API (v1.1) – a request to make this part of the full API has been made.

The response :

```
[{
        "stationId": "XX:Y:12345678",
        "name": "Island Central",
        "coordinates": {
                "lng": 6.169639,
                "lat": 52.253279
        },
        "physicalAddress": {
                "streetAddress": "example street 18, 2nd floor, 18-B33",
                "areaReference": "Smallcity, Pinetree county",
                "postalCode": "string",
                "country": "BE"
        },
        "crossStreet": "on the corner with Secondary Road"
}]
```


- **stationId**: Unique identifier for the station
- **name:** Name of the station (Should match Content-Language)
- **coordinates:** The location of the station(entrance) in LNG/LAT format)
- **physicalAddress**: The full address of the station (Should match Content-Language)
- **crossStreet**: Extra info on how to find the station. (Should match Content-Language)

GET operator/available-assets

In order to get this information we need to know the station ID of the station closest to the eHUB. This is requested from the /stations call.

Query parameters:

- **stationId**: the stationId where the assets are being requested.
- **Offset:** default: 0 (paging is supported from v1.1 – the start of result)
- **Limit:** default 20 (paging is supported from v1.1 – the maximum number of results)

The response :


```
[{
        "id": "string",
        "stationId": "string",
        "nrAvailable": 0,
        "assets": [{
                "id": "string",
                "isReserved": false,
                "isReservedFrom": "2020-12-06T15:07:14.645Z",
                "isReservedTo": "2020-12-06T15:07:14.645Z",
                "isDisabled": false,
                "overriddenProperties": {
                        "name": "string",
                        "location": {
                                "name": "string",
                                "stationId": "string",
                                "coordinates": {
                                        "lng": 6.169639,
```

```
                                        "lat": 52.253279
                                },
                                "physicalAddress": {
                                        "streetAddress": "example street 18, 2nd floor,
18-B33",
                                        "areaReference": "Smallcity, Pinetree county",
                                        "postalCode": "string",
                                        "country": "NL"
                                },
                                "extraInfo": {
                                        "additionalProp1": {}
                                }
                        },
                        "fuel": "NONE",
                        "energyLabel": "A",
                        "co2PerKm": 0,
                        "brand": "string",
                        "model": "string",
                        "travelAbroad": true,
                        "airConditioning": true,
                        "cabrio": true,
                        "colour": "string",
                        "cargo": "string",
                        "easyAccessibility": "LIFT",
                        "gears": 0,
                        "gearbox": "MANUAL",
                        "image":
"https://files.fietsersbond.nl/app/uploads/2014/10/30151126/ST2_Men_Side_CityKit-Stromer.jpg",
                        "infantSeat": true,
                        "persons": 0,
                        "pets": true,
                        "propulsion": "MUSCLE",
                        "smoking": false,
                        "stateOfCharge": 0,
                        "towingHook": true,
                        "undergroundParking": true,
                        "winterTires": true,
                        "other": "string",
                        "meta": {
                                "additionalProp1": {}
                        }
                }
        }],
        "assetClass": "AIR",
        "assetSubClass": "string",
        "sharedProperties": {
                "name": "string",
                "location": {
                        "name": "string",
                        "stopReference": [{
                                "type": "GTFS_STOP_ID",
                                "id": "string",
                                "country": "NL"
                        }],
                        "stationId": "string",
                        "coordinates": {
                                "lng": 6.169639,
                                "lat": 52.253279
                        },
                        "physicalAddress": {
                                "streetAddress": "example street 18, 2nd floor, 18-B33",
```

```
                                "areaReference": "Smallcity, Pinetree county",
                                "postalCode": "string",
                                "country": "NL"
                        },
                        "extraInfo": {
                                "additionalProp1": {}
                        }
                },
                "fuel": "NONE",
                "energyLabel": "A",
                "co2PerKm": 0,
                "brand": "string",
                "model": "string",
                "buildingYear": 0,
                "travelAbroad": true,
                "airConditioning": true,
                "cabrio": true,
                "colour": "string",
                "cargo": "string",
                "easyAccessibility": "LIFT",
                "gears": 0,
                "gearbox": "MANUAL",
                "image":
"https://files.fietsersbond.nl/app/uploads/2014/10/30151126/ST2_Men_Side_CityKit-Stromer.jpg",
                "infantSeat": true,
                "persons": 0,
                "pets": true,
                "propulsion": "MUSCLE",
                "smoking": true,
                "stateOfCharge": 0,
                "towingHook": true,
                "undergroundParking": true,
                "winterTires": true,
                "other": "string",
                "meta": {
                        "additionalProp1": {}
                }
        }
}]
```


- **Id:** Unique ID to identify this result
- **stationId**: Unique identifier for the station
- **nrAvailable**: Total number of assets available at the station
- **assets**: A description of the assets
    - **Id**: A unique identifier for the asset
    - **isReserved**: A boolan to indicate if the asset is reserved or not
    - **isReservedFrom**: If reserved a full date with timezone
    - **isReservedTo**: If reserved a full date with timezone
    - **isDisabled:** Is the asset in use (fe. In case of maintenance)
    - **rentalUrl:** a direct URL where the asset can be booked.
    - **overriddenProperties:** A list of properties that are different than the ones provided in the sharedProperties field for the station
- **assetClass**: The asset class, derived from the NETEX standard.
  AIR, BUS, TROLLEYBUS, TRAM, COACH, RAIL, INTERCITYRAIL, URBANRAIL, METRO,

`WATER, CABLEWAY, FUNICULAR, TAXI, SELFDRIVE, FOOT, BICYCLE, MOTORCYCLE, CAR, SHUTTLE, OTHER, PARKING, MOPED, STEP`

- **assetSubClass**: In the case of OTHER as assetClass – specify the class here.
- **sharedProperties**: a list of properties shared among the available assets at the station

Headers

For both incoming and outgoing requests, certain header parameters are required and are important for the handling of the request.

Incoming request headers:

```
Accept-Language: nl, en;q=0.8 … (A comma-separated list of BCP 47 (RFC 5646) language tags and optional weights as described in IETF RFC7231 section 5.3.5)
Api: TOMP
Api-Version: 1.1 (The version of the TOMP-API you have implemented. 1.1 is the latest version at the time of this document).
```

Outgoing request headers:

```
Content-Language: nl ( a BCP 47 (RFC 5646) language tag )
```

All language dependant content should be served in the language served in this tag.

Error handling

In all forms of digital communication, error handling is one of the keystones for success.

It is pertinent to use the correct HTTP error codes as described in section 10 of RFC2616: https://tools.ietf.org/html/rfc2616#section-10

A typical error response body is to be implemented as follows:

```
{ "errorcode": 0,
  "type": "string",
  "title": "string",
  "status": 0,
  "detail": "string"}
```

- **errorcode**: a TOMP specific error code, see below for details.
- **type**: Type of error

- **title**: a short human-readable description of the error, should be in the language specified in the Content-Language header.
- **status**: The HTTP status code ([RFC7231], Section 6)
- **detail**: a human-readable long(er) description of the problem, should be in the language specified in the Content-Language header.

Possible error codes for the eKA:

Code	Endpoint	Type	Title	Description
1001	operator	Missing	Field: {}, Reason: {}	
2002	plannings			
1002	operator	Invalid	Field: {}, Reason: {}	
2002	plannings			
1004	operator	Illegal operation	Operation {} is illegal in current status.	
2004	plannings			
1005	operator	Technical issue	Internal technical problem, contact support.	
2005	plannings			
1006	operator	Technical issue	No access to the endpoint	Using the authentication provided, this endpoint cannot be used.
2006	plannings			
1007	operator	Technical issue	Request limit	You've reached the maximum amount of requests per time period.
2007	plannings			
1008	operator	Technical issue	Unsupported API-version	The version of the API you're trying to use is not supported.
2008	plannings			

Table 3: TOMP-API error codes overview.

c. Compliancy with the TOMP-API definition

The TOMP-API workgroup is working on defining the compliancy levels for TOMP-API integration based on the Sarasini model explained in chapter 2.

The eHUBS kiosk API falls in Level 1: Integration of information.

For implementation purposes, we don't strictly require the endpoints – GET *regions*, GET *operating-calendar*, GET *operation-hours* and GET *pricing-plans.*
It is however recommended to follow the standard guidelines (for reusability by other partners

to start with), therefore it is advised to provide this information by the endpoints stated. This is mainly static information and does not require great effort on the PT's part.

The image below depicts the different levels of implementation and the endpoints related to each compliance level as laid out by the TOMP-WG when writing this document.

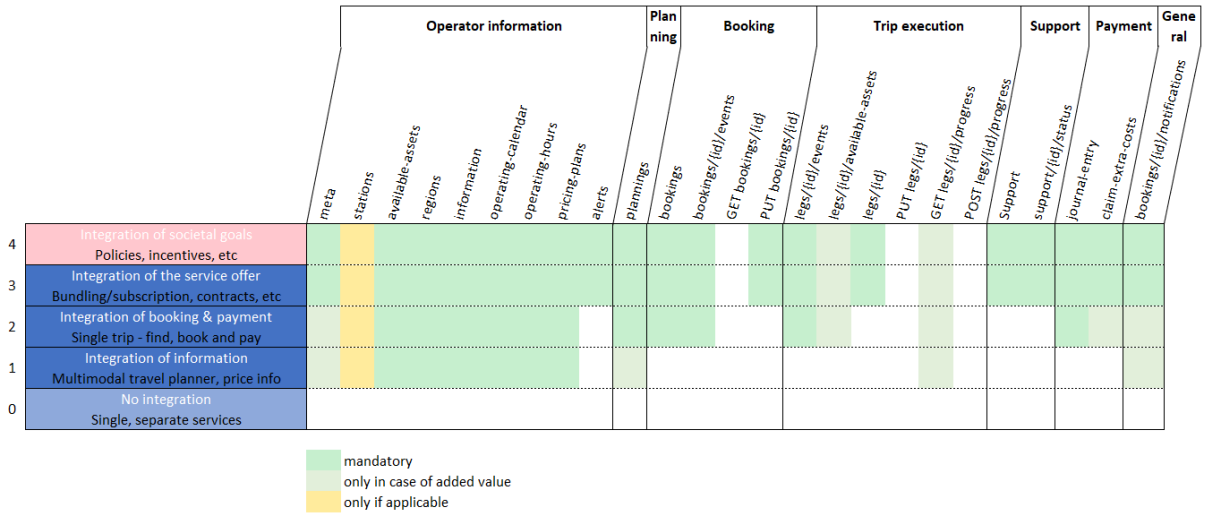

Figure 4: The compliancy levels for Transport operators in the TOMP-API – TOMP working group.

The TOMP-API documentation is the favoured source of information on the specific endpoints and the implementation thereof.

6. Authentication

Each connection and webservice call needs to be authenticated to ensure a safe transfer of data. The level and method of security are dependent on the internal structure and preferences of each company. Therefore we cannot make one form an obligation.
The 3 methods we plan to support are:

Basic Authentication:

The most basic form of authentication built into the HTTP protocol. Each HTTP request is sent with an Authorization header. This header value contains the word Basic followed by a space and a base64 encoded string containing (username: password).
The result looks like follows:

```
Basic: dXNlcm5hbWU6IHBhc3N3b3Jk
```

Basic Authentication should only be used over an HTTPS/SSL connection because the base64

encoding can easily be decoded. Communication over the HTTPS protocol prevents external parties to hijack the communication to read this header.

oAuth 2.0

OAuth 2.0 is a standard for handling authentication among web-enabled devices and servers. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for different types of applications and devices.

This method of login is used by Google, Facebook, Twitter to connect to their API's and to enable third party login for many websites.

OAuth 2.0 is the recommended method of authentication by the TOMP-WG.

JWT Webtoken

JWT web tokens is an open standard for authentication where server-generated JSON objects (tokens), valid for a short period in time, are exchanged and used for verification of access. This information can be verified and trusted because it is digitally signed.

A JWT token contains the following information:

1. **Header**: This part contains the metadata, containing the signing method and the encryption method applied to the token.
2. **Payload**: Contains information about the authenticated user.
3. **Signature**: The last part is of course very important, by signing the token you verify that what is stated above is true and unaltered.

This whole token gets base64 encoded, therefore this should always be sent over a secure HTTPS/SLL connection.

More information: https://jwt.io/introduction/

7. The KIOSK

The information KIOSK application aims to inform users about the intermodal mobility offerings available near the eHUB location. The application can be accessed from a digital eHUB pillar from a touchscreen interface.
The image below de gives an overview of the looks and some functionalities.

Figure 5: a collection of screenshots/designs – IPR (BE)

While not in use the passing people see an image carousel with room for city marketing and general information on the eHUB.
Once a user engages with the pillar a dashboard appears, providing a variety of mobility offerings. Depending on the location of course a variety of Mobility resources is shown.
This can range from Public Transport (Trains, Buses,…) to shared mobility offerings (car- & bike-sharing, e-scooters,…)
Additionally, there is room for information on the location, the vicinity, the surroundings. This can be in the form of a map, text or touristic information.
The schema below depicts the different available options.

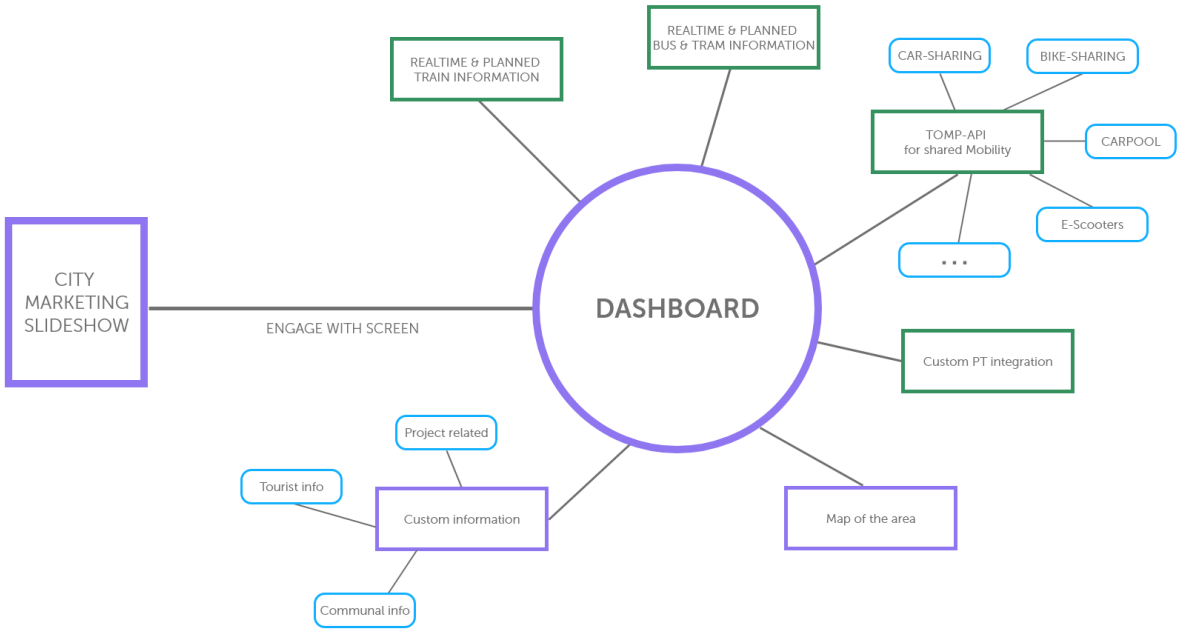

Figure 6: a schematic overview of the KIOSK application

8. The eHUBS Consortium

The consortium of eHUBS consists of 15 partners with multidisciplinary and complementary competencies. This includes European cities, leading universities, networks and electric and shared mobility providers.

@eHUBS_NWE
#eHUBS
https://www.linkedin.com/groups/13711468/

For further information please visit http://www.nweurope.eu/ehubs