Converting static data from excel to the datastandaard database using the API

# Goal

Contractors Trajan and GroenLicht are busy defining the static data of the Prorail survey, which will count parked bikes in the vicinity of Dutch railway stations.

Currently the areas are being defined. You will receive them in Excel sheets combined with shape files. The first goal is to create a tool that allows users to upload these excel sheets and polygons and will post their contents to the API.

Adding data falls apart in the following steps:

Static data:

1. Add Survey Areas by uploading a excel or csv file
2. Add Parking Locations by uploading a excel or csv file
3. Add Sections by uploading a excel or csv file
4. Add Canonical Vehicles by posting a form
5. Add Surveys by posting a form

Dynamic data:

Specifications will follow…

Feel free to add questions and remarks to this document, and correct mistakes.

To give you an impression of what we are aiming at, an extremely simple wireframe-like example can be found at <https://haverweb.nl/crow/tool/>. Its only purpose is to communicate more to-the-point about the functionality of the tool. Design is totally up to you.

# The Excel sheet

The excel sheet contains several sub-sheets, for each entity one sheet and some for the relations between entities. Those sub-sheets must be uploaded separately. So you’ll have to break up the original excel sheet into several small ones.

The upload-format is up to you: excel or csv. CSV-files will be the easiest way, I guess.

An example of the excel-sheet can be found at

<https://docs.google.com/spreadsheets/d/1uKImMAKKqYzx0zV1iP1Fln411yYjilvT/edit#gid=1292068051>

# API definition (and Swagger)

The API is located at <https://remote.veiligstallenontwikkel.nl/rest/api/v2/>

This is the test environment, so feel free to experiment.

Each entity has a secret DELETE endpoint which deletes all items. This may come at hand while developing your tool.

So to delete all parking-facilities, call

DELETE - start in the order below

[https://remote.veiligstallenontwikkel.nl/rest/api/v2/sections/](https://remote.veiligstallenontwikkel.nl/rest/api/v2/)

[https://remote.veiligstallenontwikkel.nl/rest/api/v2/parking-locations/](https://remote.veiligstallenontwikkel.nl/api/v2/)

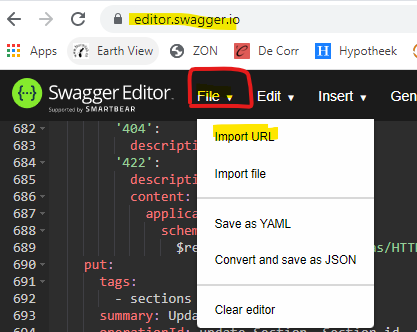
[https://remote.veiligstallenontwikkel.nl/rest/api/v2/survey-areas/](https://remote.veiligstallenontwikkel.nl/api/v2/)

The endpoints are defined in a yaml-file. You can find this file here: <https://raw.githubusercontent.com/Stichting-CROW/datastandaard-fietsparkeren/gh-pages/docs/api/assets/openapi.yaml>

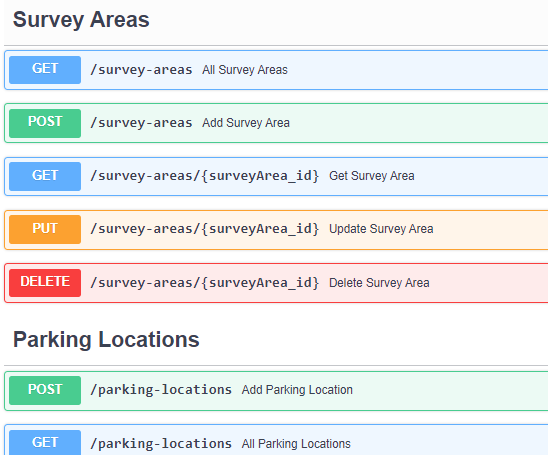
Open this file in a local swagger editor or online at:

<https://editor.swagger.io/>

File > Import URL > copy the given url



Resulting in:



For the time being any endpoint, even the GETs, require a **basic http authentication**. Use

**trajan / 9079**

and

**groenlicht / 5201**

for posting static data.

The authenticated contractor’s id is stored to any static entity that is posted. This will give this contractor and this contractor only permission to alter or delete the entity.

So:

IMPORTANT: use the Trajan credentials to upload the static data provided by Trajan

and use the Trajan credentials to upload the static data provided by GroenLicht.

# Organisations

Organisations are already present in the database. They can’t be added or altered. That’s the reason there are only GET endpoints present.

To obtain all organisations, use

GET <https://remote.veiligstallenontwikkel.nl/rest/api/v2/organisations>

Most of them are the towns in The Netherlands. They have the role ‘authority’, which doesn’t give them for the time being just view permissions. There’s also an authority ‘Prorail’, the maintainer of the Dutch railways. That’ll be the authority to use in your POSTs of static data.

Some of the organisations have the ‘contractor’ role. Those organisations are allowed to post static and dynamic data. For now, it will be only static data.

For instance:

GET <https://remote.veiligstallenontwikkel.nl/rest/api/v2/organisations/trajan>

{

"name": "Trajan",

"id": "trajan",

"roles": [

"contractor"

]

}

The excel sheet

You will be provided with an excel-sheet containing several sheets:



Each sheet contains the data of one of the following entities:

* Survey areas
* Parking Facilities
* Sections

If there are more sheets present inside the main sheet, ignore them for the time being.

# Survey Areas

Survey Areas are geographical parts in the vicinity of a railway station

Survey Areas may contain other Survey Areas. For instance : a street, a square, etc.

# Parking Locations

Parking locations are locations designed to park bikes. They must be part of a Survey Area.

# Sections

Sections are parts of a Parking Facility, such as a bicycle rack.

# GeoLocation

All static entities have a property called geoLocation. GeoLocations will be Polygons or Multi-Polygons. You can find them in the GIS shape file. They probably map on property ‘localID’, but I’m not sure about that.

By the way: I didn’t succeed to define them correctly in the openapi-file. But they must have the format explained in:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/shape.html>

It will be challenging to format them correctly. There might be tools to convert the database format provided in the excel sheet into the json-format. Good luck!

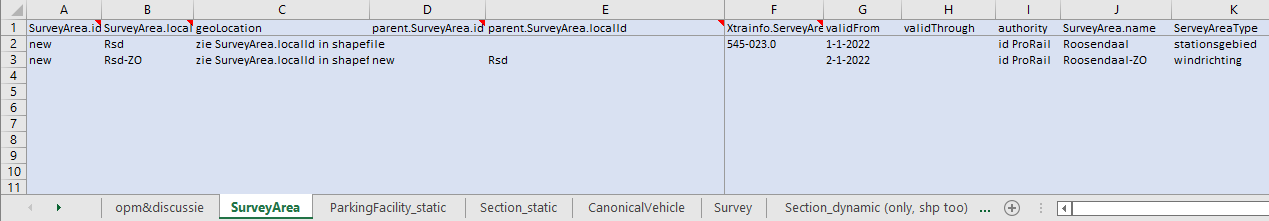
# Adding data

## Survey Areas

API endpoint:

POST <https://remote.veiligstallenontwikkel.nl/rest/api/v2/survey-areas>

Open the sheet SurveyArea of the excel sheet:



Start with the Survey Areas without parents. You’ll have to post them first and keep their id’s. Those id’s must be used to fill the parent-property of the children.

So in the above example, the first row is a parent.

This line translates in an object like this, to be posted on

POST <https://remote.veiligstallenontwikkel.nl/rest/api/v2/survey-areas> :

{

"authority": "prorail",

"name": "Roosendaal",

"localId": "Rsd",

"geoLocation": {

"type": "MultiPolygon",

"coordinates": [

[

[

[ long, lat ], [ ], [ ], [ ], [ ], [ ]

],

[

[ ], [ ], [ ], [ ], [ ], [ ]

]

]

]

},

"validFrom": "2022-01-01T00:00:00",

"surveyAreaType": "stationsgebied"

}

The response will be the entire object, with an id:

{

"authority": "prorail",

"name": "Roosendaal",

"localId": "Rsd",

"geoLocation": {

"coordinates": …,

"type": "MultiPolygon"

},

"validFrom": "2022-01-01T00:00:00",

"surveyAreaType": "stationsgebied",

"id": "D841B587-D37B-4CCD-8BF02B5B69706DE3"

}

The second row is a child of the first survey area. So paste the parent’s id in property ‘parent’ like this:

POST <https://remote.veiligstallenontwikkel.nl/rest/api/v2/survey-areas> :

{

"parent": "D841B587-D37B-4CCD-8BF02B5B69706DE3",

"authority": "prorail",

"name": "Roosendaal",

"localId": "Rsd",

"geoLocation": { … },

"validFrom": "2022-01-01T00:00:00"

"surveyAreaType": "windrichting"

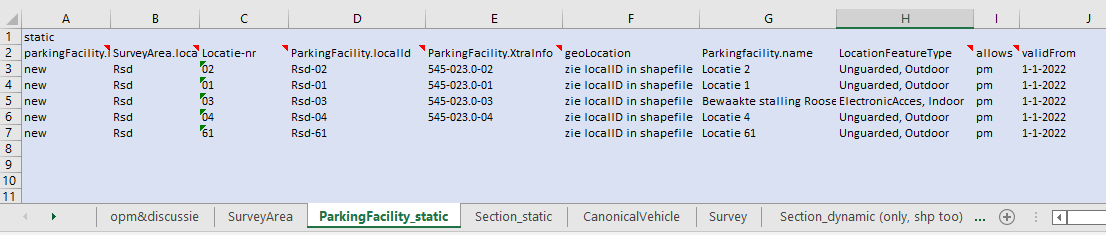
}

## Parking Locations

API endpoint:

POST <https://remote.veiligstallenontwikkel.nl/rest/api/v2/parking-locations>

Open the sheet SurveyArea of the excel sheet:



Posting parking facilities is similar to posting survey areas, except for the absence of nested entities: parking facilities do not have children.

An example post:

The first line of the example sheet translates to this object:

Column ‘LocationFeatureTypes’ must be converted to the property ‘features’, which contains an array of enum:

- lockerService

- personnelSupervision

- electronicAccess

- indoor

- outdoor

- surfaceParking

- unguarded

Thus, the first row in the sheet translates to this object:

{

"authority": "prorail",

"name": "Locatie 2",

"localId": "Rsd-02",

"geoLocation": {...},

"features": ["unguarded","outdoor"],

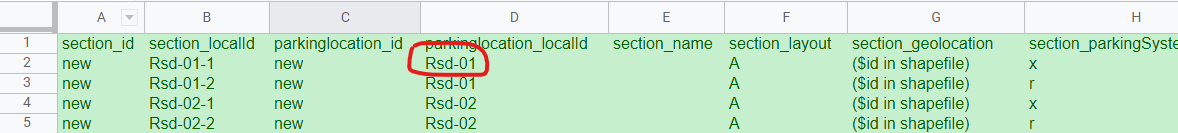
"validFrom": "2022-01-01T00:00:00"

}

## Sections

API endpoint:

POST <https://remote.veiligstallenontwikkel.nl/rest/api/v2/sections>



The column parkingLocation\_localId and parkingLocation\_authority map to a ParkingLocation. Find this ParkingLocation by querying the API like this:

parking-locations?authority=prorail&localId=Rsd-01

Grab the parkings location’s id and fill it in property Section.parkingLocation.

The first row translates into this object:

{

"parkingLocation": <parking-location-ID>,

"authority": "prorail",

"name": "",

"layout": "A",

"parkingSystemType": "x",

"localId": "Rsd-01-1",

"level": 0,

"validFrom": "2022-01-01T00:00:00"

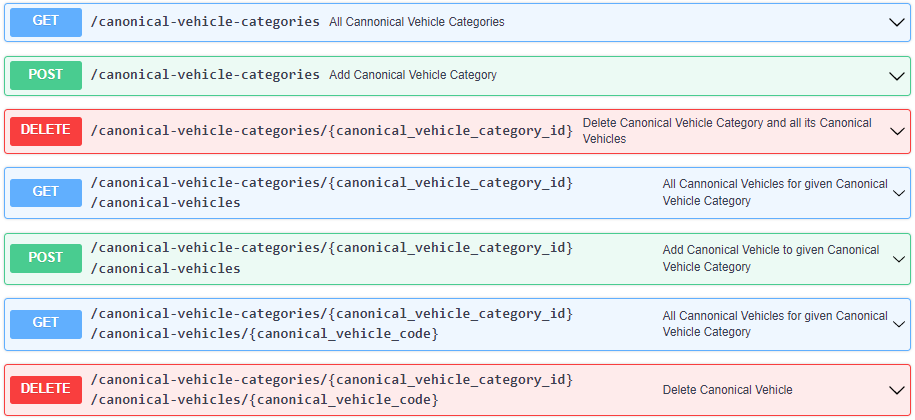
}

## Canonical Vehicles

User of the tool must be able to add so-called Canonical Vehicles. A Canonical Vehicle is a predefined bicycle. This will make the counting of bikes easier. A bike-counter can just count bikes of type ‘X’ instead of ‘scooter, improperly parked’.

Canonical Vehicles (short CV) are grouped by Canonical Vehicle Categories (short: CVC). One CVC has multiple CVs. Within a CVC each CV has a unique property ‘code’.

The following API-endpoints are available to add and read CVCs and CVs:



As you can see it’s not possible to edit (PUT) a CVC or CV.

I’m not sure if both DELETE-endpoints are allowed (I will ask CROW), but for developing purposes they may come at hand.

A Canonical Vehicle Category has got two properties:

{

"id": "Prorailcategorisering2022",

"authority": "prorail"

}

A Canonical Vehicle has the following properties:

{

"name": "standaard (e)fietsen",

"description": "alle (elektrische) fietsen behalve B, buiten voorziening",

"code": "X",

"json": {

"vehicle": {

"type": "f",

"appearance": "-x"

},

"parkState": "x"

}

}

The number of CVCs and CVs will be limited, so there’s no need for a csv-upload. A form will do:

CVC:

id: Required Text field, maxlength = 255

authority: Required Text field, maxlength = 255

CV

category: Required select with all the CVCs as options

description: Optional Textarea

code: Required Text field, maxlength = 4

json: Required Textarea, with initial value:

{

"vehicle": {

"type": "",

"appearance": ""

},

"parkState": ""

}

Value for prop ‘json’ must be a valid json-object.

## Surveys

A survey is the root object which binds static objects like survey-areas to actual bike counts. It also contains some meta-data of the survey itself



### Posting Surveys

A survey has got the following properties:

{

"name": "Prorail railway station counts 2022",

"authority": "prorail",

"contractor": "groenlicht",

"sectionLayout": "A",

"canonicalVehicleCategory": "Prorailcategorisering2022",

"license": "openbaar"

}

Use

POST <https://remote.veiligstallenontwikkel.nl/rest/api/v2/surveys>

to store a survey.

On success the complete object will be returned, including an *id.* You’ll need this id to connect survey areas to the survey.

### Form

As the number of Surveys will be limited, I suggest we offer a **form** to create new surveys:

Name: Required Text field, maxlength = 255

Authority: Required Text field, maxlength = 255

Contractor: Required Text field, maxlength = 255

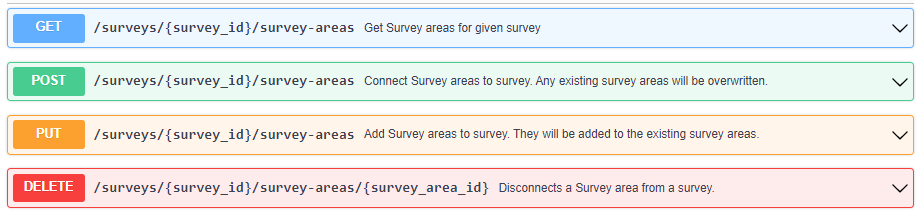
SectionLayout: Contractor: Required Text field, maxlength = 1

DistinguishesVehicleCategories: Select-box with unique values of *CanonicalVehicleCategoryIds* (to be obtained by GET <https://remote.veiligstallenontwikkel.nl//rest/api/v2/canonical-vehicle-categories/> )

License: Select-box with, for the time being, only one option: ‘openbaar’ (which means ‘public’)

## Connecting Survey-Areas to Survey

After a survey has been created, it’s time to connect the survey-areas to.



To connect a survey area, use the post-endpoint.

POST [https://remote.veiligstallenontwikkel.nl/rest/api/v2/surveys/:surveyID/survey-areas](https://remote.veiligstallenontwikkel.nl/rest/api/v2/surveys/:suveyID/survey-areas)

[

"8AD5DD72-80CF-4B72-94E071AD1E145664",

"8AD5DD72-80CF-4B72-94E071AD1E145665",

"8AD5DD72-80CF-4B72-94E071AD1E145666"

]

The body contains an array of survey-area-IDs. Any non-existing or duplicate survey-area-IDs will be ignored.

**Important note: Using this post will completely replace the existing survey-areas!**

To add a survey area to the existing set, use PUT:

PUT <https://remote.veiligstallenontwikkel.nl/rest/api/v2/surveys/:suveyID/survey-areas>

To remove a survey area from a survey, use DELETE:

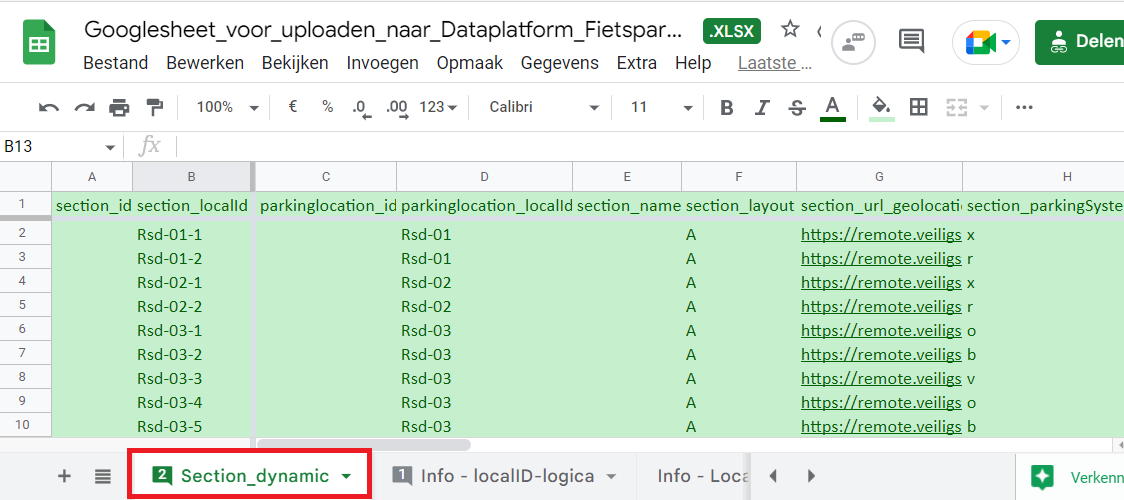
DELETE <https://remote.veiligstallenontwikkel.nl/rest/api/v2/surveys/:suveyID/survey-areas/:survey-areaID>

# Download Counting Sheets

Once the static data has been uploaded and Survey Areas have been connected to Surveys the actual bike count can begin.

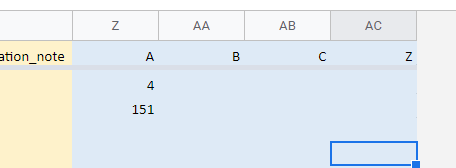
The first step in this process is to download Counting Sheets. This is a csv file with a row for every Section in the Survey.

The tab ‘Section\_dynamic’ in the sheet shows how this sheet looks like.



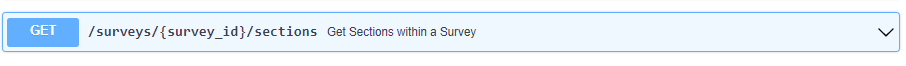
The columns in green must be filled, all other columns can be left empty.

The blue columns at the very end of a row correspond to all Canonical Vehicles connected to the Survey. For each CV there must be a column with the **code** of the CV in their headers.

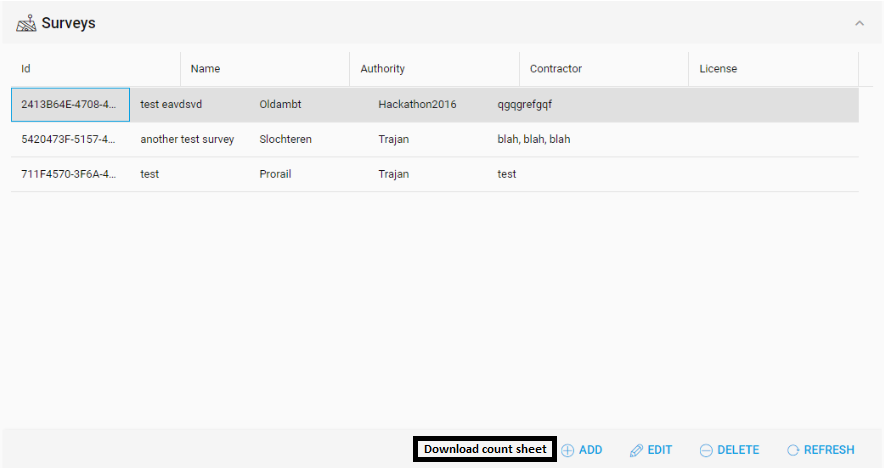


To obtain all Sections within a Survey use the newly created endpoint:

GET /surveys/{survey\_id}/sections



To let the user download the sheet, add a new button ‘Download telsheet’ on the Survey page.



## Mapping

This is the mapping between objects and the columns in the csv counting sheets:

Get sections for Survey:

<https://remote.veiligstallenontwikkel.nl/rest/api/v2/surveys/><[survey.id>/sections](http://survey.id/)

Create a row for each Section:

section\_id: <section.id>

section\_localId: <section.localid>

parkinglocation\_id: <section.parkingLocation>

parkinglocation\_localId: <parkingLocation.localid>

section\_name: <section.name>

section\_layout: <section.layout>

section\_url\_geolocation: https://remote.veiligstallenontwikkel.nl/rest/api/v2/sections/<section.id>

section\_parkingSystemType: <section.parkingSystemType>

section\_vehicleOwnerType: <section.vehicleOwnerType>

section\_level: <section.level>

section\_validFrom: <section.validFrom>

section\_validThrough: <section.validThrough>

survey\_id: <survey.id>

contractor: <survey.contractor>

observation\_capacity\_id: <empty>

observation\_capacity\_timestamp\_start: <empty>

observation\_capacity\_timestamp\_end: <empty>

capacity\_parkingCapacity: <empty>

observation\_capacity\_note: <empty>

observation\_occupation\_timestamp\_start: <empty>

observation\_occupation\_timestamp\_end: <empty>

occupation\_totalParked: <empty>

observation\_occupation\_note: <empty>

Get Canonical Vehicles for survey:

[https://remote.veiligstallenontwikkel.nl//rest/api/v2/canonical-vehicle-categories/<survey.canonicalVehicleCategory>/canonical-vehicles](https://remote.veiligstallenontwikkel.nl//rest/api/v2/canonical-vehicle-categories/%7Bsurvey.canonicalVehicleCategory%7D/canonical-vehicles)

Create a column for each Canonical Vehicle

<canonicalVehicle.code>: <empty>