

Helsingin kaupunki

Helsinki Region Infoshare Converter Tool Documentation

JSON data in CSV

Contents

About	2
Overview	2
Proof of concept	2
Technical Description	3
CSV –format	5
Datsette	6
Extending to other APIs	9
Data Usability	10

About

Author: Anders Innovations Oy, Toni Pellinen

This document describes the purpose and functionality of the HRI API Publisher –program.

Overview

The program is intended to offer an alternative way of using the data from the open APIs of the City of Helsinki. This is done by creating and offering user-friendly CSV -files available for download. These CSVs contain the same data as the API, but in a restructured format that allows usage with Excel for example.

Proof of concept

Proof of concept source code available here for testing:

<https://github.com/City-of-Helsinki/hri-api-publisher>

Other links:

City of Helsinki Servicemap

<https://palvelukartta.hel.fi/fi/>

Servicemap API on hri.fi

<https://hri.fi/data/fi/dataset/paakaupunkiseudun-palvelukartan-rest-rajapinta>

Servicemap API Documentation

<https://dev.hel.fi/apis/service-map-backend-api>

Technical Description

The program has two main functionalities: Pulling the data from API, restructuring it and saving it to a database, and creating a web interface with Datasette, where user can download CSV -files.

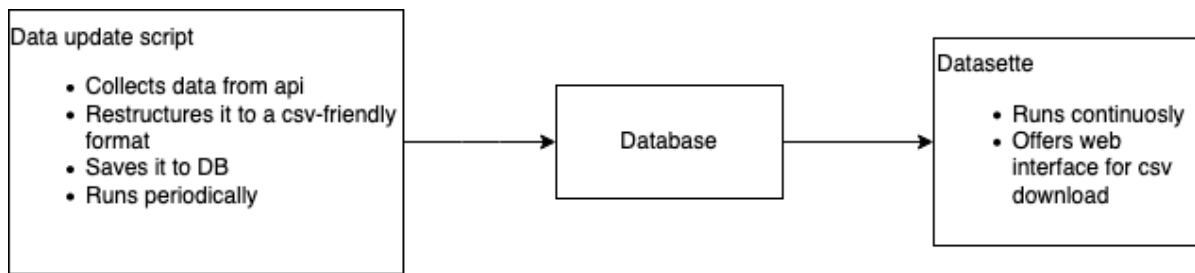
Program is designed to have the Datasette running continuously, and data updating can happen with set intervals. Data updating updates the data to the same database as Datasette uses, so there is no down-time in the web interface.

Data updating has three main steps. First, it pulls all the data from the API (ignoring possible pre-defined endpoints). This is the bottleneck in the program, and to speed up the execution time, requests are done asynchronously with 20 concurrent requests.

Second step is to restructure data into CSV -friendly format. This is done by “flattening” the json-format, so that every nested object or list field in the json is removed and the data is restructured from those fields to the original json’s root. This is optional step, if some fields do not need flattening, they can be ignored.

Last step is to save the data to the database. This program uses SQLite as a database.

Program flowchart:



Minimal example how the JSON –restructuring works:

```
[
  {
    "name": {
      "fi": "Helsingin Seurakuntayhtymä",
      "sv": "Helsingfors Kyrkliga Samfällighet"
    },
    "list_of_items": [
      2356,
      452765
    ]
  },
  {
    "name_fi": "Helsingin Seurakuntayhtymä",
    "name_sv": "Helsingfors Kyrkliga Samfällighet",
    "list_of_items_0": 2356,
    "list_of_items_1": 452765
  }
]
```

Note how the nested object and list fields are converted to the first level with suffixes, such as “_fi” and “_0”.

CSV –format

CSV -files have all the data from the API in a restructured format, where all list –fields and nested objects are converted to the json root. This restructuring allows using the CSV –files with Excel for example. However, if the mentioned list –fields are very long or the nested objects that are more than one level deep, the CSV might become cluttered with data, which might decrease usability. This can be avoided by specifying these fields to be ignored in flattening – phase.

Currently data is split into multiple CSVs, where each endpoint in a single API has their own file. This causes small usability issues, because data may be split into multiple files. Excel at least has some tools to combine these files based on the field values, and it could also be built into the application. It might decrease usability though, if the data file becomes very long or contains too many fields.

CSVs use comma as a separator.

Datasette

Datasette is used to download the CSV-files. Here is an example picture from the front page, which shows all the endpoints available in the application. First section is custom SQL query, which allows more flexible customization of the data to download for the user.

The screenshot shows the Datasette web interface. At the top, there is a dark blue header with the word "home" in white. Below the header, the word "servicemap" is displayed in a large, bold, black font. Underneath, the section "Custom SQL query" is visible. A text input field contains the SQL query: `1 select * from address`. Below the input field, there are two buttons: "Format SQL" and "Run SQL". The results of the query are displayed below the buttons, showing a list of database tables with their respective columns and row counts:

- address**: `id, number, number_end, letter, modified_at, location_type, location_coordinates_0, location_coordinates_1, street_id...` 126,220 rows
- administrative division**: `id, origin_id, ocd_id, service_point_id, start, end, modified_at, type, parent...` 2,680 rows
- administrative division type**: `id, type, name` 21 rows
- announcement**: `id, title_fi, title_sv, title_en, lead_paragraph_fi, lead_paragraph_sv, lead_paragraph_en, content_fi, content_sv...` 1 row

Customized query to get only three columns from Address – endpoint:

servicemap

Custom SQL query returning more than 1,000 rows ([hide](#))

```
1 select id, number, street_name_fi from address
```

Format SQL

Run SQL

This data as [json](#), [CSV](#)

id	number	street_name_fi
1	1	Adolf Lindforsin tie
2	11	Adolf Lindforsin tie
3	2	Adolf Lindforsin tie
4	3	Adolf Lindforsin tie
5	5	Adolf Lindforsin tie
6	7	Adolf Lindforsin tie
7	9	Adolf Lindforsin tie

In the Address-page there is alternative filtering options, whole data table, and possibility to download the CSV:

address

126,220 rows

- column - =

Apply

[View and edit SQL](#)

This data as [json](#), [CSV](#) ([advanced](#))

Suggested facets: [modified_at](#) (date), [street_modified_at](#) (date)

id	number	number_end	letter	modified_at	location_type
1	1			2018-08-07T16:12:26.139794+03:00	Point
2	11			2018-08-07T16:12:26.139952+03:00	Point
3	2			2018-08-07T16:12:37.280409+03:00	Point
4	3			2018-08-07T16:12:26.140080+03:00	Point
5	5			2018-08-07T16:12:37.280542+03:00	Point

Simple filtering options without SQL:

address

50,406 rows where street_municipality = "helsinki" sorted by id

street_municipality = helsinki

- column - =

Apply

User can also modify the SQL query here:

home / servicemap

servicemap

Custom SQL query returning 101 rows ([hide](#))

```
1 select id, number, number_end, letter, modified_at, location_type, location_coordinates_0, location_coordinates_1, street_id, street_modified_at, street_municipality, street_name_fi, street_name_sv, postal_code from address order by id limit 101
```

Format SQL Run SQL

In the end of the page is the download option. Both checkboxes need

to be checked when exporting the CSV:

Advanced export

JSON shape: [default](#), [array](#), [newline-delimited](#), [object](#)

CSV options: **download file** **stream all rows** [Export CSV](#)

```
CREATE TABLE [address] (  
  [id] INTEGER PRIMARY KEY,  
  [number] TEXT,  
  [number_end] TEXT,  
  [letter] TEXT,  
  [modified_at] TEXT,  
  [location_type] TEXT,  
  [location_coordinates_0] FLOAT,  
  [location_coordinates_1] FLOAT,  
  [street_id] INTEGER,  
  [street_modified_at] TEXT,  
  [street_municipality] TEXT,  
  [street_name_fi] TEXT,  
  [street_name_sv] TEXT,  
  [postal_code] TEXT  
);
```

Powered by [Datasette](#) · Query took 1440.731ms

Extending to other APIs

This program can be extended to cover other APIs of the City of Helsinki. When extending, the predefined environment variables need to be set, such as the API URL, database name, endpoints that are ignored and API fields that are saved as json to the CSV. In addition, the structure of the new API needs to be the same, or otherwise the program code needs some restructuring. For example, the linked events –API has API metadata under the field called “meta”, whereas in service map the metadata is as root values. Program would expect them currently as root values, so the path would need to be fixed for the linked events –API.

Data Usability

The generated CSV files are usable with basic tools such as excel.

Filtering out unwanted data might require some problem solving skills but should be doable without a technical background using the basic functionalities of excel or similar tools.

Hiding or deleting unwanted columns and filtering by recurring keywords makes the data quite usable for most purposes.

	B	E	AQ	
1	connections_0_section_type	connections_0_name_en	name_en	street_address_fi
8	OPENING_HOURS	open Mon-Fri 09-16:30	Playground Piika	Arentikuja 5
9	OPENING_HOURS	open Mon-Fri 09-16	Playground Kaunokki	Tuohipolku 10
12	OPENING_HOURS	open Mon-Fri 09-16:30	Playground Maunula	Kuusikkotie 2a
18	OPENING_HOURS	open Mon-Fri 09-16	Playground Mäkitorppa	Mäkitorpantie 42
51	OPENING_HOURS		Playground ja perhetalo Torpparinmäki	Ylätuvanpolku 4
53	OPENING_HOURS	open Mon-Fri 09-16	Playground Soihtu	Ylipalontie 1
61	OPENING_HOURS	open Mon-Fri 09-16:30	Playground Lahnahti	Lauttasaarentie 40a
63	OPENING_HOURS	open Mon-Fri 09-16:30	Playground Ruoholahti	Laivapojankatu 8
98	OPENING_HOURS	open Mon-Fri 09:30-16:30	Playground Lehdokki	Kasöörinkatu 3 (sisätila) Pakkamestarinkatu 4 (ulkopuisto)
107	OPENING_HOURS	open Mon-Tue 09:30-16:30, Wed-Thu 09-16:30	Playground Kimmo	Turjantie 3
108	OPENING_HOURS	open Mon, Wed, Fri 09:30-16:30, Tue, Thu 09-16:30	Playground Brahe	Porvoonkatu 4
110	OPENING_HOURS	open Mon-Fri 09-16:30	Playground Linja	Toinen linja 6
112	OPENING_HOURS	open Mon-Fri 09:30-16:30	Playground Intia	Intiankatu 44
115	OPENING_HOURS	open Mon-Fri 09-16	Playground Vallila	Anjalantie 1 C
117	OPENING_HOURS	open Mon-Tue 09-16:30, Wed-Fri 09-16:30	Playground Sanna	Pasilan puistotie 8

Some fields however are dependent of other endpoints and are not human readable. As an example, from the Service map unit endpoint the accessibility data only points to other data found in a different endpoint.

	AQ	CZ
1	name_en	accessibility_properties_32_variable
8	Playground Piika	43
9	Playground Kaunokki	44
12	Playground Maunula	
18	Playground Mäkitorppa	
51	Playground ja perhetalo Torpparinmäki	
53	Playground Soihtu	
61	Playground Lahnahti	20
63	Playground Ruoholahti	50
98	Playground Lehdokki	50

To make this data usable it would have to be combined with the data of the other endpoint to turn it into a readable form, this fell outside of the scope for the proof of concept. Matching the id's and variables to the corresponding data would most likely have to be done manually to each API the conversion software would be implemented to.