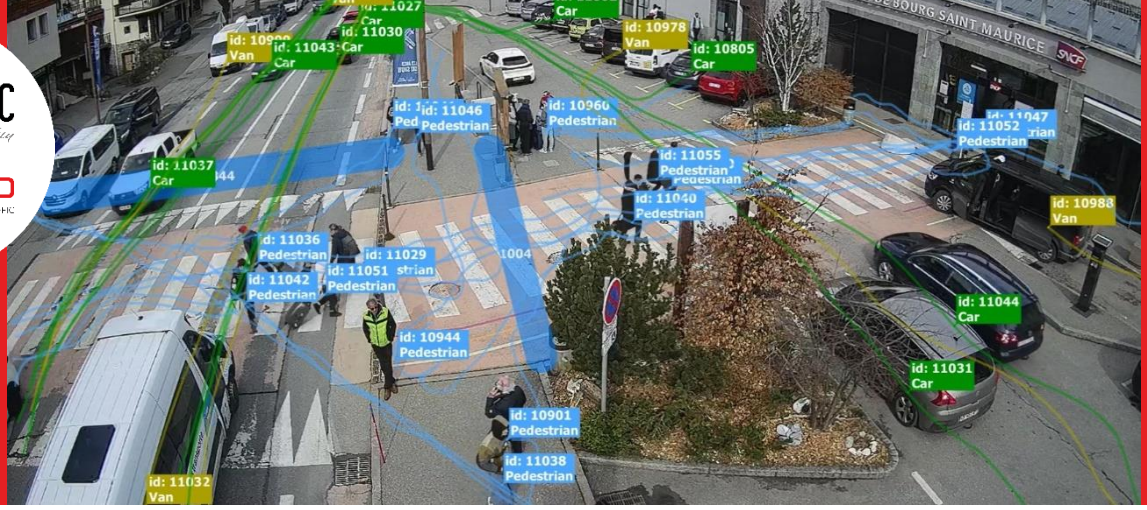


Overview of sink types

See the various details of each sink type.

Operator attributes and sink types

There are many forms of information you can get about traffic data passing through a certain operator. You may want statistical data about speed or acceleration of all passing traffic, or maybe a speed heatmap. From the standpoint of the server's HTTP response, the sink's **type** denotes the contents of each sink's data object – the names and data types of its member variables. It also denotes additional parameters that may be a part of sink output requests. The **attribute** that the sink is attached to decides the values of these members. The following table shows a list of all attributes that all operators have along with their associated sink type.

Filter attribute	Sink type
Object count	Value
Level of service	Value
Speed	Statistical value
Stationary time	Statistical value
Duration of occurrence	Statistical value
Trajectory list	Table
Trajectory view	Trajectory view
Category distribution	Distribution
Color distribution	Distribution
Speed map	Heatmap
Acceleration map	Heatmap
Occupancy map	Heatmap
OD matrix	OD matrix
Raw trajectory data	Raw trajectories

Light vs. heavy data

The sinks are categorized as *light data* or *heavy data* sinks. This reflects how large their data responses tend to be in general. To prevent sink data responses from growing out of proportion, it's only possible to request outputs from one heavy data sink at a time. In other words, when requesting for a heavy data sink output, no other light data or heavy data sink output can be requested.

Details of sink types

In this section, you will find detailed descriptions of each sink type. The properties data_validity, evaluation_validity, and object_count are common for all sink types. They're all explained in the previous article.

Value

JSON type identifier: value
History supported: Yes
Output heaviness: Light data
Additional request parameters: None
Response's data object contents: A single numerical value
Example of response's data contents:

JSON


```
"data": {
  "data_validity": "ok",
  "evaluation_validity":"ok",
  "object_count": 57,
  "value": 57
}
```


Statistical value

JSON type identifier: statistical_value
History supported: Yes
Output heaviness: Light data
Additional request parameters: None
Response's data object contents: A collection of key/value pairs. The values are always a single number. The keys designate: Minimum value, maximum value, median, average, mode, variance, and size (number of samples). Size is currently redundant and has the same value as object_count.
Example of response's data contents: JSON


```
"data": {
  "data_validity": "ok",
  "evaluation_validity":"ok",
  "object_count": 624,
  "minimum": 0.0,
  "maximum": 51.6,
  "median":  41.1,
  "average": 40.2,
  "size": 624
}
```

Table

Tables are currently limited to hold up to 10,000 records of closed trajectories. There's no limit for additional open trajectories. Sending all of them at once can be demanding on the hardware and bandwidth. That's why each response will hold a maximum of 100 table records. In order to obtain further rows, you need to include further parameters in your data request. By default, tables are ordered by the "Trajectory start" column in ascending order.

JSON type identifier: table
History supported: No
Output heaviness: Heavy data
Additional request parameters:

- number_of_rows - maximum number of rows you want to receive in one page. The actual received number can be lower if there are not enough rows in the table, or if you provided a number greater than 100.
- first_row_index - the row index at which the returned page shall start
- order_by - the name of column by which you want the page to be ordered. You can see column names in the response example below in the header array. The table is ordered *before* a page is extracted.
- order - either ascending, or descending

(All of these parameters are optional.)

Example request body:

JSON

```
{
  "sequence_number": "11",
  "sinks": [
    {
      "id": 0,
      "number_of_rows": 100,
      "first_row_index": 200,
      "order_by": "Maximum speed",
      "order": "descending"
    }
  ]
}
```

Response's data object contents:

- table_size - total number of rows in the table; currently redundant and synonymous with object_count
- number_of_rows - number of rows in the current page
- header - an array of strings designating the names of table columns. You can read more about the Gap time, Gap distance, and Section speed colums in a dedicated article.
- units - the measurement units used in each column. kpx are kilopixels. Positions and sizes are in normalized screen coordinates - values from 0.0 to 1.0 where 0.0 is in the upper left corner of the image.
- formats - the data types and formats of each column. The float type also states the number of decimal places. The datetime type specifies a format according to Qt's format strings for date and time.
- data_start - the value of the column of the first row in the returned page, always as a string. If the table is ordered by "Trajectory start", then this property is instead named data_start_time.
- data_end - the value of the column of the last row in the returned page, always as a string. If the table is ordered by "Trajectory start", then this property is instead named data_end_time.
- order - either ascending, or descending
- data - an array of arrays, each containing one table row

See the Trajectory view sink for a list of possible values in the Category column.

Example of response's data contents: JSON

"data": {

"data_validity": "ok",

"evaluation_validity": "ok",

"object_count": 16,

"data_end_time": "1618991959204",

"data_start_time": "1618990159105",

"first_row_index": 0,

"number_of_rows": 16,

"order": "ascending",

"table_size": 16,

```
 "header": [ "ID", "License plate", "Category", "Color",
             "Trajectory start", "Trajectory end", "Average speed",
             "Minimum speed", "Maximum speed", "Section speed",
             "Average acceleration", "Gap time", "Gap distance",
             "Duration of occurrence", "Stationary duration",
             "Last position X", "Last position Y",
             "Last bounding box X", "Last bounding box Y",
             "Last bounding box width", "Last bounding box height" ],
 "units": [ "", "", "", "", "ms", "ms", "kpx/h", "kpx/h",
            "kpx/h", "km/h", "px/s²", "s", "px", "s", "s",
            "", "", "", "", "", "" ],
 "formats": [ "int", "string", "string", "string",
              "datetime;yyyy-MM-dd hh:mm:ss",
              "datetime;yyyy-MM-dd hh:mm:ss",
              "float;0", "float;0", "float;0", "float;0",
              "float;1", "float;2", "float;0", "float;2",
                                    "float;2", "float;6", "float;6", "float;6",
                                    "float;6", "float;6", "float;6" ],
 "data": [
   [ "5", "Undefined", "pedestrian", "undefined", "1618990434162",
     "1618990442603", "162.32", "13.31", "719.34", "-", "0.97",
     "-", "-", "54.59", "31.40", "0.984672", "0.768475",
     "0.984672", "0.768475", "0.014019", "0.069893" ],
   [ "8", "Undefined", "car", "white", "1685370299216",
     "1685370338389", "173.54", "26.60", "581.06", "-", "1.55",
     "-", "-", "39.17", "18.48", "0.133255", "0.104881",
     "0.134138", "0.103832", "0.055505", "0.073010" ],
   // ...
 ]
}
```

Distribution

JSON type identifier: distribution
History supported: Yes
Output heaviness: Light data
Additional request parameters: None
Response's data object contents: A collection of key/value pairs, where keys are histogram categories (classes), and values are single numerical values. The possible categories depend on swissTRAFFIC AI node's current configuration. You can find them out with the Camera information request.
Example of response's data contents for a category distribution :

JSON


```
"data": {
  "data_validity": "ok",
  "evaluation_validity":"ok",
  "object_count": 130,
  "categories": [
    {
      "category": "bicycle",
      "count": 13
    },
    {
      "category": "bus",
      "count": 3
    },
    {
      "category": "car",
      "count": 74
    },
    {
      "category": "light",
      "count": 11
    },
```

```
      {
        "category": "pedestrian",
        "count": 29
      },
      {
        "category": "unknown",
        "count": 0
      }
   ]
}
```


Example of response’s data contents for a color distribution:

JSON


```
"data": {
   "data_validity": "ok",
   "evaluation_validity":"ok",
   "object_count": 14,
   "colors": [
      {
        "color": "black",
        "count": 4
      },
      {
        "color": "blue",
        "count": 0
      },
      {
        "color": "brown",
        "count": 0
      },
      {
```

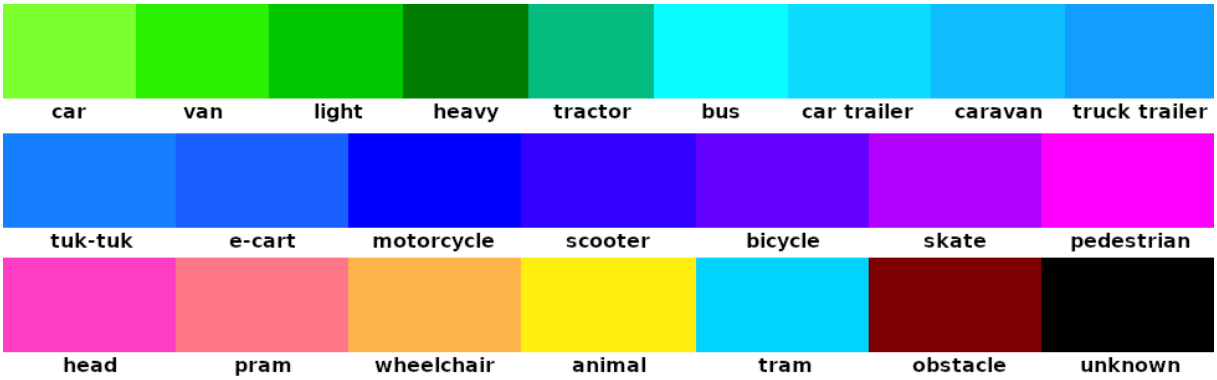
```
      "color": "red",
      "count": 1
    },
    {
      "color": "silver",
      "count": 2
    },
    {
      "color": "undefined",
      "count": 7
    }
  ]
}
```


Trajectory view

JSON type identifier: trajectory_view
History supported: No
Output heaviness: Heavy data
Additional request parameters: None
Response's data object contents:

- rendered - denotes whether the outputs are PNG images, or raw data. This will always be set to true.
- rendered_trajectories - an array of objects, which contain:
    - data - a string containing a Base64 representation of a PNG image. This image is a visualization of all selected trajectories of the corresponding category over a transparent background.
    - category - a string denoting a category whose trajectories are rendered in the corresponding image. The possible values and the corresponding rendered color are:
        - car - #7CFF2E
        - van - #2BF000
        - light (Light truck) - #00C800
        - heavy (Heavy truck) - #007D00
        - tractor - #04BC80

- bus - #07FBFF
- car trailer - #0BDCFF
- caravan - #0FBDFF
- truck trailer - #129EFF
- tuk-tuk - #167FFF
- e-cart - #195FFF
- motorcycle - #0000FF
- scooter - #3200FF
- bicycle - #6400FF
- skate - #B200FF
- pedestrian - #FF00FF
- head - #FF3DC3
- pram - #FF7787
- wheelchair - #FFB34B
- animal - #FFEE0F
- tram - #00D3FE
- obstacle - #7F0000
- unknown - #000000



- If there are no objects to be rendered for a given category, or the category is not supported by the swissTRAFFIC AI node, the whole object will be omitted in the rendered_trajectories array. E.g. in the example below, categories bus, motorcycle, and e-cart have been omitted and the other categories are not supported.

Example of response's data contents:

JSON


```
"data": {
  "data_validity": "ok",
  "evaluation_validity": "ok",
  "object_count": 17,
  "timestamp": "1619697039715",
  "rendered": true,
  "rendered_trajectories": [
    {
      "category": "unknown",
      "data": "iVBORw0KGgoAAAANSUhEUgAA", // ...
    },
    {
      "category": "car",
      "data": "iVBORw0KGgoAAAANSUhEUgAA", // ...
    },
    {
      "category": "light",
      "data": "iVBORw0KGgoAAAANSUhEUgAA", // ...
    },
    {
      "category": "heavy",
      "data": "iVBORw0KGgoAAAANSUhEUgAA", // ...
    },
    {
      "category": "bicycle",
      "data": "iVBORw0KGgoAAAANSUhEUgAA", // ...
    },
    {
```

```
      "category": "pedestrian",
      "data": "iVBORw0KGgoAAAANSUhEUgAA", // ...
    }
  ]
}
```

Example of all trajectory view images overlaid over a video snapshot:

Comments:

- It might take a long time for a block to respond to a trajectory view sink data request. Due to this, all trajectory view sink data requests must be sent separately - without requests for any other sink data in the same message.

Heatmap

JSON type identifier: heatmap
History supported: No
Output heaviness: Heavy data
Additional request parameters: None
Response's data object contents: A map_type property, which can have either the Heatmap, or Gridmap value. This section will concern the Heatmap type - Gridmap will be described in the next one. The response also contains two matrices: sum_data and count_data. A quick guide to parse the matrix data:

1. Decode the string from Base64.
2. Unzip with the ZLIB library's flate algorithm.

3. Parse the first four 32-bit integers representing the number of rows, number of columns, OpenCV element type (always 32-bit float), and number of channels (always 1) respectively.
4. Parse the 1D array of 32-bit floats representing the matrix data.

The matrices are represented as a raw 1D array of 32-bit floats, but the matrices are of the same total size as the video image (i.e. the array's size is image_height*image_width). Each element is a value that relates to the corresponding pixel in the original image - elements of the sum_data matrix represent a sum of a value selected during the sink's creation (e. g. current speed) and elements of the count_data matrix represent the number of samples.

For example, for an image with resolution 1920x1080 and **Speed map** type of heatmap, the second element of sum_data would contain a sum of speeds of all trajectories that crossed the second pixel in the first row, and the second element of the count_data matrix would contain the number of those trajectories. Therefore, by dividing the sum value by the count value, you can obtain the average value for the corresponding pixel in the input image.

For another example, if the image has a resolution of 1920x1080 px, then the matrices would be represented as 1D arrays containing 1920*1080 = 2073600 elements. Element 68372 of the array would correspond to the image pixel in a row floor (68372/1920) = 35 and column 68372%1920 = 1172 where % represents the modulus operation.

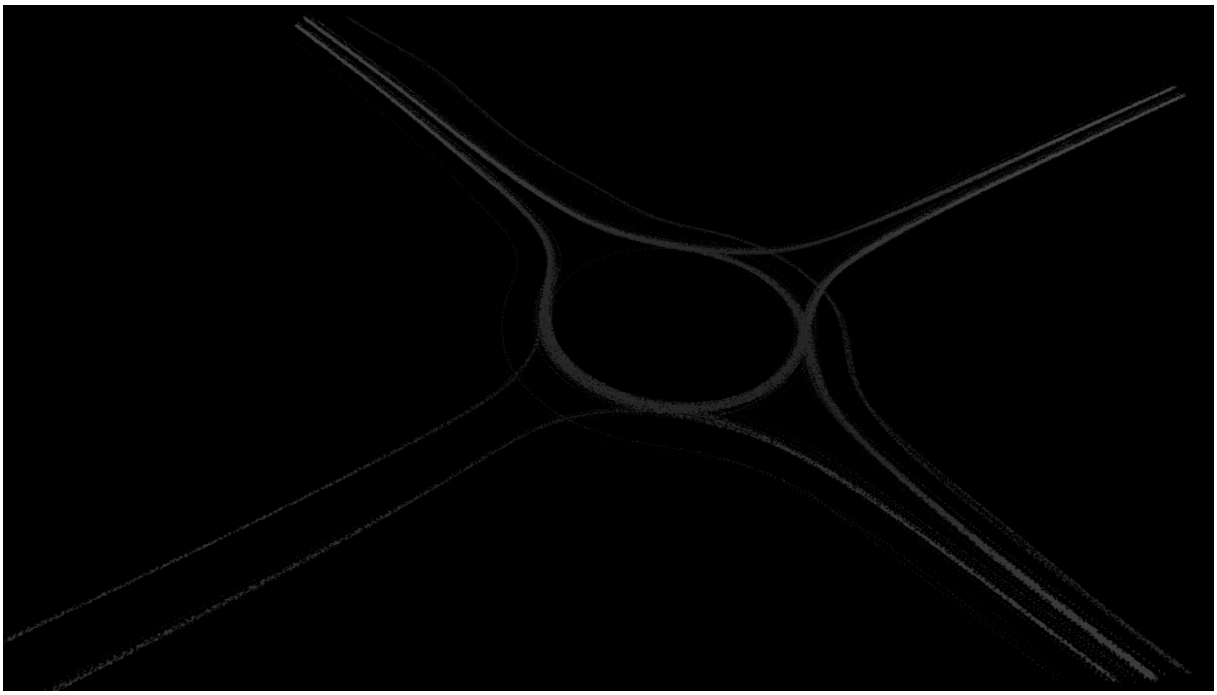
The matrices are quite large, so each of them is compressed with the ZIP (deflate) algorithm of the ZLIB library and encoded in Base64 before being placed into JSON response and sent.

Example of response's data contents:

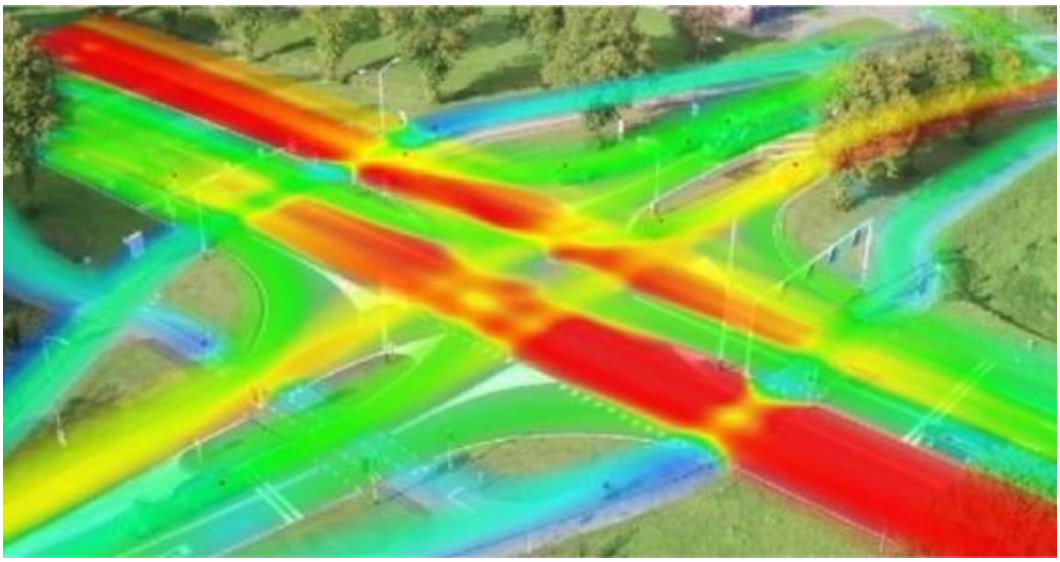
JSON


```
"data": {
  "object_count": 420,
  "data_validity": "ok",
  "evaluation_validity":"ok",
  "map_type": "Heatmap",
  "count_data": "TWFuIGlzIGRpc3RpbmXNoZWQsIG5vdCB5IGhpcyByZWF", // ...
  "sum_data": "WQsIG5vdCBvbmx5IGJ5IGhpcyByZWd1aXNoZWQsIG55as", // ...
  "timestamp": "1685008212409"
}
```

Example of a heatmap converted to a grayscale image:



For illustration purposes, here is an example of a processed (colored) heatmap overlaid over a corresponding video snapshot:



Comments:

- timestamp has the same meaning and value as the sink's timestamp value outside of the data object, which is described in Getting the list of all sinks

Gridmap

Technically, gridmap is a subtype of the heatmap, but its output is a bit different, so it'll be explained here in a separate section. See the explanation of a gridmap widget to learn what it is. It's the same with a gridmap sink.

JSON type identifier: heatmap
History supported: No
Output heaviness: Heavy data
Additional request parameters: None
Response's data object contents: A map_type property, which can have either the Heatmap, or Gridmap value. This section will concern the Gridmap type - Heatmap is described in the previous one. Then there is tile_size, which denotes how many pixels are aggregated in one grid tile in each dimension. For example, a tile_size of 50 means that each grid aggregates 50x50=2500 pixels. Moreover, for an image of size 1920x1080, the grid has ceil(1920/50)=39 columns and ceil(1080/50)=22 rows. (ceil() is an operation of rounding up to the nearest integer.) Finally, the response's data object contains another data property, which contains a JSON string with the encoded grid. A quick guide to decode it:

1. Decode the string from Base64.
2. Unzip with the ZLIB library's flate algorithm.
3. Parse the array of 32-bit floats representing the matrix data.

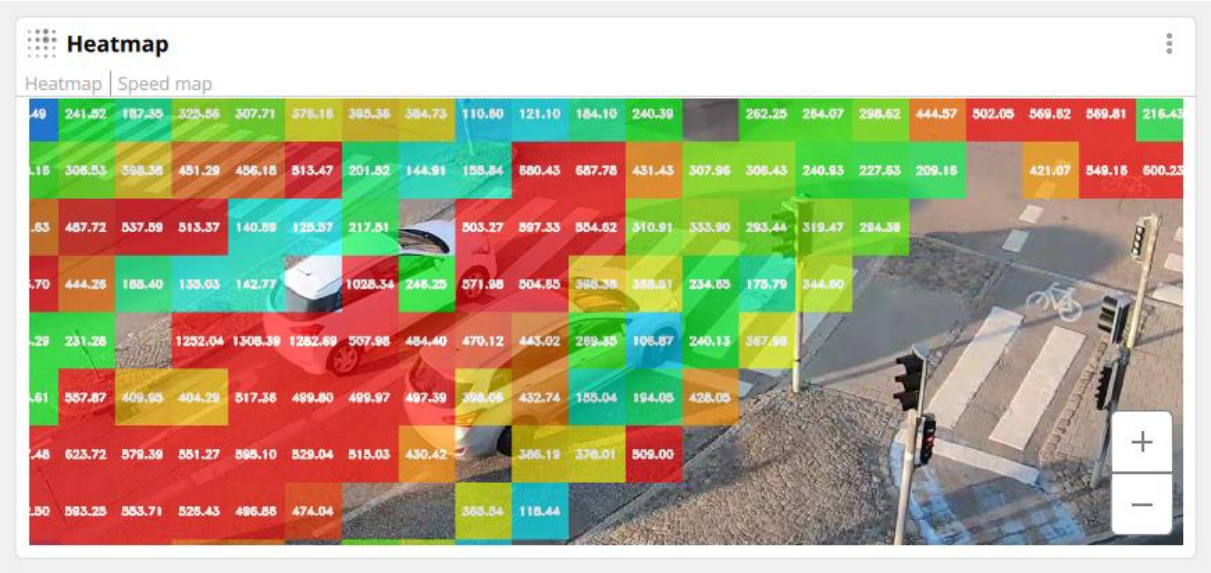
Tiles are stored in a row-major order. Each tile comprises of 5 32-bit floats: Minimum, maximum, average, median, and object count. These statistical values are tied to the operator attribute that's been set for the sink: speed, acceleration, object count etc.

Example of response's data contents:

JSON

```
"data": {
  "object_count": 420,
  "data_validity": "ok",
  "evaluation_validity": "ok",
  "map_type": "Gridmap",
  "source_height": 1520,
  "source_width": 2704,
  "tile_size": 50,
  "data": "eJzsnHlQFVcWhzsKQUFxQ1xxgR" // ...
}
```

Example of a gridmap displayed as a widget in swissTRAFFIC AI Insights:



OD matrix

OD matrix functionality is explained in a dedicated how-to article.

JSON type identifier: od_matrix
History supported: Yes (in all time modes), the number of records is **limited to 1000**
Output heaviness: Light data
Additional request parameters: None
Response's data object contents:

- origins - an array of entry gates (user name and ID)
- destinations - an array of exit gates (user name and ID)
- turning_movements - rows of turning counts

The data have a row-oriented format where the rows represent entry/origin gates. The columns are exit/destination gates. The order of the elements in the headers (origins/destinations) corresponds to the order of the data in the turning_movements section. Maximum size is 8x8.

Example of response's data contents:

(the size of the OD matrix is 1x2 i.e. 1 entry and 2 exits)

JSON

```
"data": {
    "data_validity": "ok",
    "evaluation_validity": "ok",
    "object_count": 604,
    "origins": [            // OD origins/entries
```

```
    {
      "id": 1,          // id of the entry gate
      "name": "In"      // name of the entry gate
    }
  ],
  "destinations": [         // OD destinations/exits
    {
      "id": 3,          // id of the exit gate (generated by swissTRAFFIC AI)
      "name": "Out 1"   // name of the gate (defined by user)
    },
    {
      "id": 4,
      "name": "Out 2"
    }
  ],
  "turning_movements": [   // data is entry/row-oriented
    {
      "category": "car",
      "data": [
        [ 0, 5025]  // In -> Out 1: 0; In -> Out 2: 5025
      ]
    },
    {
      "category": "pedestrian",
      "data": [
        [ 0, 19]  // In -> Out 1: 0; In -> Out 2: 19
      ]
    },
    // ...the rest of all supported categories
  ]
```

```
  ]
}
```

Raw trajectories

This sink shows you the evolution of traffic objects' trajectories.

JSON type identifier: raw_trajectories
History supported: No
Output heaviness: Heavy data
Additional request parameters: None
Response's data object contents:

JSON

```
{
        "data_validity": "ok",
  "evaluation_validity": "ok",
        "object_count": 1329,
        "trajectories": [
    {
      "color": "white",
      "duration": 39142,
      "id": "8",
      "license_plate": "",
      "last_detection_score": 0.5966202020645142,
      "trajectory_score": 0.8626661896705627,
      "timestamp": "1650529144653",
      "category": "car",
      "state_data": {
        "timestamps": [
          0,
          368,
          // ...
        ],
        "map_positions": [
```

```
  [615951.5625, 5453988.5],
  [615951.8125, 5453992],
  // ...
],
"map_speeds": [
  0.09022672474384308,
  0.021549025550484657,
  // ...
],
"map_accelerations": [
  5.666559219360352,
  -10.65473175048828,
  // ...
],
"map_speed_angles": [
  5.109283447265625,
  5.166502952575684,
  // ...
],
"wgs84_positions": [
  [16.59258449695335, 49.22769412627738],
  [16.592588941067113, 49.22772555017891],
  // ...
],
"sensor_positions": [
  [ 0.11871673911809921, 0.1282760351896286 ],
  [ 0.12449279427528381, 0.12761197984218597 ],
  // ...
],
"bounding_boxes": [
```

```
        [
          0.25482144951820374, 0.2587209641933441,
          0.27610158920288086, 0.29117339849472046
        ],
        [
          0.2526055872440338, 0.2627129852771759,
          0.28182610869407654, 0.29768529534339905
        ],
        // ...
      ]
    }
                },
  // ...
        ]
}
```


Comments:

- trajectories - information about the traffic objects outputted by the sink
    - color - the general color of the traffic object
    - duration - how long the traffic object has been in the scene
    - id - the traffic object's unique identifier within the analytics
    - license_plate - the license plate contents of the traffic object. If none have been detected, the value is an empty string.
    - timestamp - the timestamp of the moment when this object has been detected for the first time in milliseconds since epoch
    - category - the traffic object's category
    - last_detection_score - the most recent detection confidence of the traffic object normalized to the interval 0.0-1.0
    - trajectory_score - the average of the last several detection confidence scores normalized to the interval 0.0-1.0
    - state_data - data about the traffic object's properties that change with time

- map_positions - only available in georegistered analytics. This array contains the traffic object's positions within its corresponding UTM zone. The first element of the position array is the *x* coordinate, the second element is the *y* coordinate.
- map_speeds - only available in georegistered analytics. This array contains the traffic object's speeds in m/s.
- map_accelerations - only available in georegistered analytics. This array contains the traffic object's acceleration in m/s^2.
- map_speed_angles - only available in georegistered analytics. This array contains the traffic object's azimuths within its corresponding UTM zone in radians, with the value 0 denoting direction to the right (along the x axis).
- timestamps - the amount of milliseconds that has passed between the moment when this object has been detected for the first time and the moment that the other state_data properties correspond to.
- wgs84_positions - the position of the traffic object within the WGS 84 coordinate system. Only available in georegistered analytics. The first array element is the longitude, the second is latitude. They're expressed as positive or negative numbers. Positive latitude is above the equator (N), and negative latitude is below the equator (S). Positive longitude is east of the prime meridian, while negative longitude is west of the prime meridian (a north-south line that runs through a point in England).
- sensor_positions - the position of the traffic object within the sensor's normalized coordinate space. The first array element is the x coordinate, the second is the y coordinate. E.g. in a camera image, the coordinates [0,0] correspond to the upper left corner.
- sensor_speeds - only available in non-georegistered analytics. This array contains the traffic object's speeds in px/s.
- sensor_acceleration - only available in non-georegistered analytics. This array contains the traffic object's accelerations in px/s^2.
- sensor_speed_angles - only available in non-georegistered analytics. This array contains the traffic object's orientation within the sensor coordinate space in radians, with the value 0 denoting direction to the right (along the x axis).
- bounding_boxes - this array contains sub-arrays, where each sub-array denotes an object's bounding box properties using 4 doubles with semantics in the following order: top left point's x coordinate, top left's y, bottom right's x, bottom right's y. The coordinates are normalized screen coordinates (values from 0 to 1 with [0,0] being in the upper left corner).

Trajectory file sink

This is a special type of sink that lets users of swissTRAFFIC AI Insights export observed trajectories into a .tlgx file. It's not possible to interact with it through the REST API - requesting data of this sink returns an empty result.

JSON type identifier: file

Measurement units guide

What units do different sinks and attributes give you?

Sink type	Attribute	Unit
Value	Object count	Pure number (unitless)
	Level of service	Pure number (unitless)
Statistical value	Duration of occurrence	[m/s]
	Stationary duration	[m/s]
	Speed	[kpx/s] (kilopixels per second) if using non-georegistered footage and [m/s] for georegistered footage
Table	Trajectory list	Table has a descriptive header. All **time data** is in [s] except Trajectory start and Trajectory ends which have the [datetime] format. **Speed** in [kpx/h] if not georegistered or [km/h] when georegistered. Positions and sizes are in normalized screen coordinates - values from 0.0 to 1.0 where 0.0 is in the upper left corner of the image. For more details, see the table sink description.
Distribution	Color	Pure number (unitless)
	Category	Pure number (unitless)

Sink type	Attribute	Unit
Heatmap	Speed	Heatmap has a set of 2 values for each pixel – sum_data value and a count_data value. sum_data is the sum of speeds recorded in a given pixel and uses [kpx/s] for non-georegistered footage and [m/s] for georegistered footage count_data is the number of objects that passed through that pixel and is, therefore, a unitless value
	Acceleration	Same as with speed, but the units are [kpx/s^2] and [m/s^2].
	Occupancy	sum_data is the number of objects that passed through the pixel count_data is equal to 1 if at least one object has been detected there, otherwise, the value is 0 Both of these values are unitless.
Raw trajectories	Raw trajectory data	See the units property inside the data response.

What next?

Learn more about sinks' data history.

Need more help or have some questions? Contact us!