DIGITAAL VLAANDEREN

/// Rapport

ONBOARDING VSDS

Implementation Guideline

Versie /// 0.9

Publicatiedatum /// 05 December 2023

Vlaamse overheid

www.vlaanderen.be/digitaalvlaanderen

Auteur: Koenraad Verduyn

Datum aanmaak: 5 December 2023

Datum afdruk: 8 december 2023

Interne bestandsnaam: Implementation guideline V0.91.docx

Documenthistoriek:

Versie	Opmerking	Datum	Auteur	Status
[versie]	[opmerking]	[datum]	[auteur]	[status]

Digitaal Vlaanderen

Havenlaan 88, 1000 Brussel
+32 (0)2 553 72 02

Koningin Maria Hendrikaplein 70, 9000 Gent
+32 (0)9 276 15 00

digitaal.vlaanderen@vlaanderen.be

INHOUD

1 CONTEXT

1.1 VSDS

As the Flemish Smart Data Space, we focus on making data findable, accessible and easy to (re)use, in short, we encourage sustainable data sharing. We do this by designing technical standards and building blocks, and providing an agreement framework around them that promotes trust between the parties.

As Flemish government, we assume a connecting role between the public actors and private actors who exchange data (with each other). This should lead to an open ecosystem in which parties publish and consume data in a reliable and standardised manner. Via standards and cooperation protocols, we stimulate the development of smart-data and personal data space applications.

The Flemish Smart Data Space thus ensures standardisation of data integration. That is, we aim to minimise the customisation involved in data exchange. This allows each party in the ecosystem to focus on its core tasks (data quality, extracting insights from the data, new applications, etc.) when exchanging data. Based on good agreements and the principles of Linked Data, the same data source can be deployed for multiple use cases or multiple purposes. This way, each party gets the best possible insights from the data.

1.2 OSLO

The Flemish government wants to optimise its services and make the exchange of information smoother. But government services to citizens and entrepreneurs are supported by specialised applications from different software suppliers. There is therefore a need for an unambiguous standard: OSLO (Open Standards for Linking Organisations). The aim is to ensure more consistency, better comprehensibility and better findability of information and services. That way, everyone can use the data more easily.

Authorities at local, regional, inter-federal and European level often have to cooperate in the provision of services. In practice, a lot of data must therefore be exchanged between the various administrations. These data originate from different systems, may not be available in the same technical format and do not necessarily follow the same semantics. Without making agreements, quality data exchange becomes impossible. This underlines the importance of developing data standards. The different standards can be subdivided according to where they are in the life cycle, namely:

Recognised standards: after going through a recognition procedure, were approved by the Working Group on Data Standards and the Steering Comittee Flemish Information and ICT Policy as a standard within the Flemish government.

Candidate standards: a stable specification for the standard was developed, implementation experience is gained through a public review period and feedback from outside the thematic working group is collected.

Standards under development: have already been submitted to the Working Group on Data Standards and are being developed by a thematic working group using public working sessions.

Technical standards: the technical specifications for data exchange, they offer ways of standardised data exchange.

Vocabulary: the basis for open semantic information standards, they provide a shared conceptual framework for certain concepts with a focus on data exchange.

Application profiles: a data exchange specification for applications that fulfil a particular use case. It allows additional constraints to be imposed in addition to shared semantics, such as defining cardinalities or using certain code lists.

Implementation model: a specification for data exchange for applications that fulfil a particular use case, these models may include some of the internal data processing of specific applications. It allows additional constraints to be imposed in addition to shared semantics, such as defining cardinalities or the use of certain code lists. Its development follows the same proces as application profiles and vocabularies, but does not yet have official recognition as a process and method.

At https://data.vlaanderen.be/standaarden/, you can find the Standards Register, where you can search for the appropriate application profile. For this use case, we use the OSLO standard Traffic measurements. You can find the application profile here : https://data.vlaanderen.be/standaarden/standaard-in-ontwikkeling/applicatieprofiel-verkeersmetingen.html and the vocabulary here : https://data.vlaanderen.be/standaarden/standaard-in-ontwikkeling/vocabularium-verkeersmetingen.html. You can also find all presentations and reports of the thematic working group that took on the development of this standard.

In the context of VSDS, OSLO is the functional standard to which data complies in terms of content, without making a statement about the technical carrier. The OSLO model defines the form of the data, so that all data that conforms to this model has the same structure, right down to the properties of all fields and any possible restrictions in values (code lists). The specific structure of data for cross-sectional counts is discussed further here.

1.3 LDES

LDES stands for Linked Data Event Stream. The LDES Server is a configurable component that can be used to unlock updates of data via the LDES protocol. Within the context of the Flemish Smart Data Space project, open-source LDES components have been built to facilitate the exchange of open data. Technical documentation on these components can be found here : https://informatievlaanderen.github.io/VSDS-Tech-Docs/introduction/LDES_server. The server components themselves can be found here : https://hub.docker.com/r/ldes/ldes-server/tags, and the code is publicly available on GitHub : https://github.com/informatievlaanderen/VSDS-LDESServer4j.

In addition, open-source components have also been developed for the LDES client, which can fetch and convert these LDES streams. High-level technical documentation on the LDES client is here : https://informatievlaanderen.github.io/VSDS-Tech-Docs/introduction/LDES_client, and the components

themselves are here : https://github.com/Informatievlaanderen/VSDS-Linked-Data-Interactions. LDES client is part of a larger suite for getting started with Linked Data, namely Linked Data Interactions (LDI): https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/. Two frameworks for data pipelining are supported from LDI: Apache Nifi & Linked Data Interactions Orchestrator.

LDES is the technical medium through which data is exposed. LDES consists of components specifically designed to open up linked data in an efficient and easy way, but is generic in terms of the content of the data, which is defined in the OSLO model.

2 USE CASE TRAFFIC MEASUREMENTS – CROSS-SECTIONAL COUNTS

2.1 WHAT IS A CROSS-SECTIONAL COUNT?

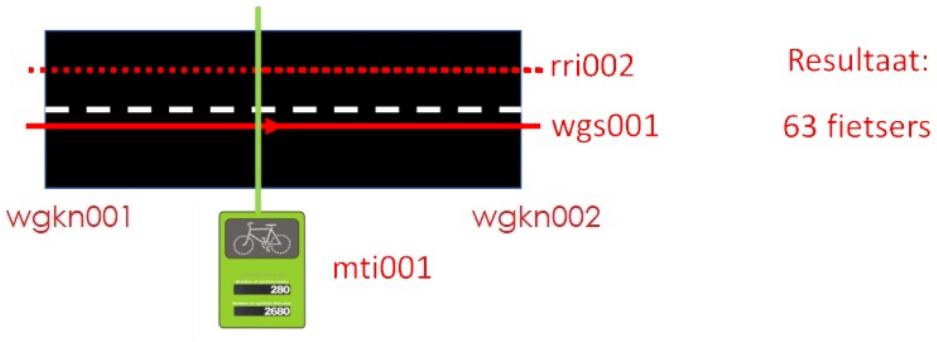
The domain of traffic measurements is a very vast domain. The use case cross-section measurements further identifies it as a measurement that meets the following conditions :

It involves a measurement at a single point in space, namely at a cross-section of a road section. Origin-destination measurements or intersection counts do not belong in this use case.

The measurement concerns a cross-section of the road, i.e. a sum of all lanes (including footpaths and/or bicycle lanes). Lane counts do not belong here.

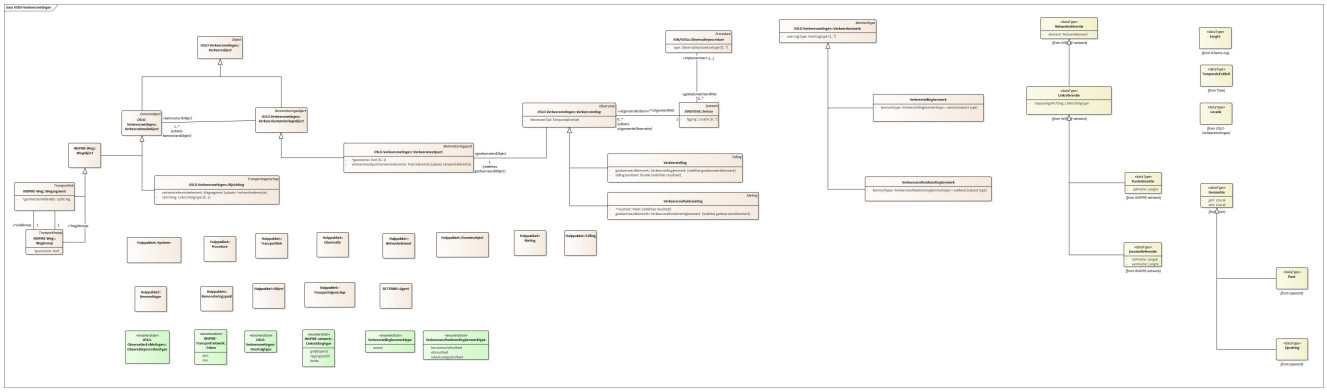
Measurements are aggregated per 15 minutes per transport mode (see below for code lists).

The measurements refer either to the number of traffic participants (count) or to the measured speed (speed measurement).



These can be either historical measurement campaigns where data is present for a certain period (in 15-minute intervals), or live counting campaigns where the data is replenished each time.

2.2 IMPLEMENTATIONMODEL



The implementation model Cross-section Counts (https://implementatie.data.test-vlaanderen.be/doc/implementatiemodel/verkeersmetingen/) is a further specification of the OSLO application profile Traffic Measurements (https://data.vlaanderen.be/doc/applicatieprofiel/verkeersmetingen/).

The implementation model does not work with a generic traffic measurement class (as in the traffic measurement application profile), but defines two classes that further specify the traffic measurement class, namely Traffic Count and Traffic Speed Count. The attribute of a Traffic Count is always of type "Number", and the result is a number of type double (decimal number).

For a traffic speed measurement, you can choose between 4 types (V85, median, time-averaged speed and place-averaged speed) as an attribute, and the unit should be included with the result. The other parts of the model are taken from the traffic measurement application profile.

The following sections contain both the input data requirements for modelling this, and an example.

2.3 DATA REQUIREMENTS

2.3.1 Counting data

At least the following data should be present :

- Location of the census (see below for specifications)
- Time/date of the count
- Vehicle type
- Counted value : this can be either speed or number
- Type of sensor

Optionally, the following can be included :

- observation procedure

ach measurement included in the LDES stream contains only 1 value for the measurement. I.e., if at 1 location both numbers and speed are measured, that becomes two measurements, one for the number of vehicles and 1 for the speed.

Further in the process, we discuss how to make the data OSLO compliant (with reference to the implementation model).

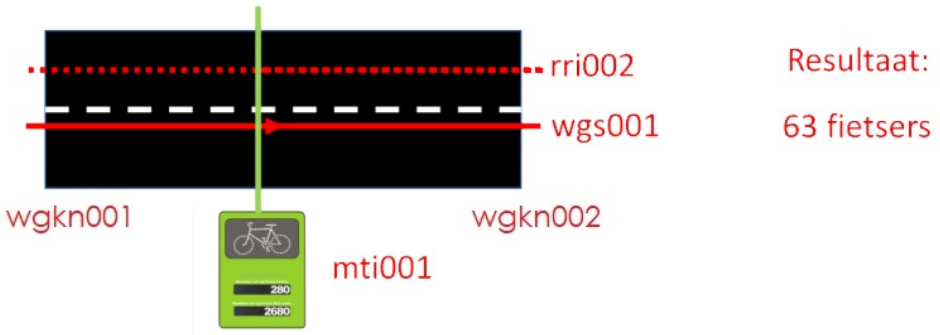
2.3.2 Location

The choice was made to store location data independently of a digital map in the model. Therefore, we follow the OpenLR specification for specifying location (http://www.openlr.org/). OpenLR was developed by TomTom to allow network-related data to be mapped onto a map without prior knowledge of the map on which it will be mapped. OpenLR was developed as open source (https://github.com/tomtom-international/openlr) under the Apache licence 2.0 (compatible with GPL v3), which allows the software to be used, distributed and modified without any royalties. The Tomtom github page lists 9 components including a decoder (which links map agnostic data back to a map).

No OpenLR software compoments are used in this process, only the specifications for location data are followed (downloadable here : http://www.openlr.org/fileadmin/user_upload/openlr-whitepaper_v1.5.pdf).

That is, two things are included : a road segment (at least consisting of 1 start node and 1 end node with XY coordinates in WGS84), and an offset in metres. This offset determines where exactly the measurement point is on the road segment with respect to the starting node.

We illustrate using a figure :



In this example, a road segment is defined WGS001, which has an A node and a B node, namely wgkn001 and wgkn002. An X and Y coordinate in WGS84 is provided for each road node. The order of this node determines the digitisation direction. That is, every other direction refers to it. Thus a driving direction (equidistant or opposite) is always relative to this digitisation direction.

The location of the measurement mti001 is then determined by passing an offset with respect to the road node A, i.e. in the example with respect to wgkn001. This offset is given in metres.

2.3.3 Time

For cross-sectional counts, we process data in 15-minute intervals. In the OSLO model, this is coded as an interval with a start and end time (or start time and duration), of the format XML schema DateTime. At least a date and a start time should be included in the input data (in 15-minute intervals). If an end duration of the interval is given, this can be adopted (in which case other intervals per measurement are technically possible, but not desirable), otherwise the interval for all counts must be given in the mapping of the data.

2.3.4 Vehicle type

For each measurement, the type of vehicle counted must be specified. The OSLO model provides a code list for this purpose that contains the following possibilities:

- Car: in addition to passenger cars, this class also includes light goods vehicles (>3.5 tonnes) and motorbikes.
- Truck : in addition to trucks, this class includes buses.
- Bicycle : this class includes classic bicycles as well as electric bicycles (including speed pedelecs) and mopeds.
- Pedestrian : in addition to pedestrians, wheelchairs and mobility scooters are included here.

If this information is not present in the input data, it must be assumed that the measured vehicles all belong to the same class, and this is assumed as a constant when mapping the data to the OSLO model.

2.3.5 Speed measurement

For a speed measurement, it is possible to choose between four types of measured speeds. The OSLO model provides a code list for this purpose that contains the following possibilities:

- V85 : the speed that 85% of the traffic does not exceed
- Median speed: the speed not exceeded by 50% of the traffic
- Locational average speed : harmonic average of the speed of all vehicles
- Time average speed : arithmetic average of the speed of all vehicles

A given value for a speed measurement must be one of these four categories. Either the type is included in the data, or it is assumed that all data are of the same category and this is included when mapping data to the OSLO model. The first option is preferred, as it obviously allows multiple types of speed to be included in the dataset.

In the OSLO model, the unit (usually km/h, sometimes m/s) should also be included. This can either be provided in the input per measurement, or specified when mapping to the OSLO model. Note that this is only possible if all speed measurements use the same unit.

2.3.6 Counts

The result of a count contains the number of vehicles. The result may be specified as a decimal number (the OSLO model expects the type "double"), which may be applicable for interpolated values, for example. For the use case cross-section counts, this is the sum of all vehicles of the specified vehicle type in the given quarter-hour.

2.3.7 Type Sensor

The type of sensor should be specified in the OSLO model. Again, the OSLO model provides a code list with the possible values :

- ANPR Camera : This is a camera that recognises and records license plates, usually for enforcement or access control purposes.
- Optical fibre : sensor that uses optical fibre
- Induction loop : sensor that detects vehicles via magnetic field changes in buried loops
- Manual count : a manually performed count
- Piezzo sensor : sensor that measures traffic pressure via piezoelectric sensors when vehicles drive over it
- Radar : sensor that emits radio waves and then detects the reflected waves to determine the position, speed and direction of objects
- Rubber hose : sensor with one or more rubber hoses for measuring traffic volume by air pressure change when vehicles are driven over
- Camera : a camera using image recognition to count vehicles without registration number registration
- Abacus : count performed by an abacus device (https://telraam.net/), a crowd sourced solution

If the sensor type is not present in the input data, the sensor type can also be specified as a constant in the mapping to the OSLO model. However, this requires that all counts were performed with the same sensor type.

2.3.8 Observationprocedure

Optionally, the implemented observation procedure can be included. In the various onboardings of cross-sectional counts, this has not proved relevant, so will not be discussed further. In the OSLO model, the specifications can be found here : https://implementatie.data.test-vlaanderen.be/doc/implementatiemodel/verkeersmetingen/#Observatieprocedure

2.3.9 Extra classes

In addition to the described code lists, it is possible to specify multiple values for vehicle type, sensor type and speed measurement. This is done by including separate array of values (so-called "double-typing") in which reference is made to a second code list and a second value from that code list. We illustrate with an example :

Firm A has the ability to distinguish between red and blue vehicles with their measurement equipment, and wants to include this level of detail in the published open data. Thus, for 1 interval, they counted 100 blue cars and 50 red cars.

In the resulting dataset, these are two separate measurements in the same time segment, namely one measurement with vehicle type "[code list VSDS :car code list Firm A:red car]" and result 50, and one measurement with vehicle type "[code list VSDS:car code list Firm A: blue car]" and result 100.

The main requirement is that both code lists are accessible. The URL to the VSDS code list is included in the standard context available with the implementation model (https://implementatie.data.test-vlaanderen.be/doc/implementatiemodel/verkeersmetingen/ontwerpstandaard/2023-09-

27/context/Verkeersmetingen-im.jsonld), the URL of the detailed code list needs to be added to the context of the data and hosted with the data owner.

There is one drawback to this method, namely that for data with very high detail, the number of measurements (and thus the number of records) increases very quickly. It is therefore recommended to monitor the impact on storage and bandwidth.

2.4 DATA EXAMPLE

In this chapter, we illustrate the transformation of input data in JSON format to an OSLO compatible JSON-LD format. We use data from the city of Bruges (available as open data) to illustrate this, but the illustrated principles apply to 90% of all traffic data available. Note : JSON-LD is chosen here purely for illustrative reasons, the LDES components write the resulting OSLO data directly to a database. JSON-LD is the most readable notation form of linked data, hence the choice.

The input data falls into two parts, namely a description of the location, and a description of the measurement. In JSON format, the location looks as follows :

```
{
    "id": "05d50e16-2e4e-414a-921a-91a1ea11cb02",
    "name": "Site 38",
    "address": "Noorweegse Kaai",
    "geometry": {
        "type": "Point",
        "coordinates": [
            3.2421013712883,
            51.2257868862067
        ]
    },
    "country": "BE",
    "subdivision": "BE-BRU",
    "origins": [
        {
            "name": "A",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    3.24239104986191,
                    51.2259355430829
                ]
            }
        },
        {
            "name": "B",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    3.24177950620651,
                    51.2256331896095
                ]
            }
        }
    ]
}
```

De eerste drie regels bevatten een ID, de naam van de site en het adres. Daar is er een blok die de XY coördinaten bevat (regel 5 t.e.m. 11).

Daarna komen er twee regels met administratieve locaties, en vervolgens het blok dat het wegsegment beschrijft waarop de telling gebeurd is (regels 14 t.e.m. 35). In dat blok is er een beginpunt beschreven (“A”), met XY coördinaat, en een eindpunt (“B”), ook met een XY coördinaat.

Na de omzetting naar een OSLO compatibel JSON-LD (linked data) ziet het resultaat (1 gecombineerd bestand met zowel locatie als telling) er als volgt uit :


```
 1. {
 2.   "@context": [
 3.
"https://implementatie.data.vlaanderen.be/doc/implementatiemodel/verkeersmetingen/ontwerpstandaard/2023-
09-27/context/Verkeersmetingen-im.jsonld",
 4.     "https://data.vlaanderen.be/doc/applicatieprofiel/sensoren-en-
bemonstering/kandidaatstandaard/2022-04-28/context/ap-sensoren-en-bemonstering.jsonld",
 5.     "https://data.vlaanderen.be/doc/applicatieprofiel/observaties-en-
metingen/kandidaatstandaard/2022-04-28/context/ap-observaties-en-metingen.jsonld",
 6.     "https://raw.githubusercontent.com/samuvack/context/main/wegenregister.jsonld",
 7.     "https://data.vlaanderen.be/doc/applicatieprofiel/GEODCAT-AP-VL/erkendestandaard/2022-04-
21/context/geodcatap-vlaanderen.jsonld",
 8.     "https://data.vlaanderen.be/doc/applicatieprofiel/generiek-basis/zonderstatus/2019-07-
01/context/generiek-basis.jsonld",
 9.     {
10.       "schema": "http://schema.org/",
11.       "dct": "http://purl.org/dc/terms/",
12.       "xsd": "http://www.w3.org/2001/XMLSchema#",
13.       "geosparql": "http://www.opengis.net/ont/geosparql#",
14.       "qudt-unit": "http://qudt.org/vocab/unit/",
15.       "qudt-schema": "https://qudt.org/schema/qudt/",
16.       "dcterms": "http://purl.org/dc/terms/",
17.       "time": "http://www.w3.org/2006/time#",
18.       "adms": "http://www.w3.org/ns/adms#",
19.       "ucum": "https://w3id.org/cdt/",
20.       "Verkeersmeting.resultaat": {
21.         "@type": "http://www.w3.org/2001/XMLSchema#double",
22.         "@id": "http://def.isotc211.org/iso19156/2011/Observation#OM_Observation.result"
23.       },
24.       "cl-vrt": "https://data.vlaanderen.be/doc/concept/VkmVoertuigType/",
25.       "cl-vkt": "https://data.vlaanderen.be/doc/concept/VkmVerkeersKenmerkType/",
26.       "cl-trt": "https://inspire.ec.europa.eu/codelist/LinkDirectionValue/",
27.       "cl-mit": "https://data.vlaanderen.be/doc/concept/VkmMeetInstrumentType/",
28.       "cl-op": "https://data.vlaanderen.be/doc/concept/VkmObservatieProcedure/",
29.       "cl-access": "http://publications.europa.eu/resource/authority/access-right/"
30.     }
31.   ],
32.   "@graph": [
33.     {
34.       "@id": "_:GM001",
35.       "@type": "Dataset",
36.       "Dataset.titel": {
37.         "@language": "nl",
38.         "@value": "GeoMobility."
39.       },
40.       "Dataset.beschrijving": [
41.         {
42.           "@language": "nl",
```

```
43.           "@value": "Floating Car Data geeft een zeer goed beeld van de verkeerssituatie in een stad,
maar soms is het essentieel om exact te weten hoeveel verkeer er in bothDirections richtingen door een
straat rijdt en wat de modal split is."
44.         }
45.       ],
46.       "Dataset.toegankelijkheid": "cl-access:PUBLIC",
47.       "Dataset.trefwoord": [
48.         {
49.           "@language": "nl",
50.           "@value": "GeoMobility"
51.         }
52.       ]
53.     },
54.     {
55.       "@id": "_:rri001",
56.       "@type": "Rijrichting",
57.       "Rijrichting.netwerkreferentieelement": {
58.         "@type": "Linkreferentie",
59.         "Linkreferentie.toepassingsRichting": "cl-trt:inDirection",
60.         "Netwerkreferentie.element": "_:wgs001"
61.       },
62.       "Rijrichting.rijrichting": "cl-trt:inDirection"
63.     },
64.     {
65.       "@id": "_:wgs001",
66.       "@type": "Wegsegment",
67.       "Wegsegment.beginknoop": "_:wgkn001",
68.       "Wegsegment.eindknoop": "_:wgkn002",
69.       "Wegsegment.geometriemiddenlijn": {
70.         "@type": "LineString",
71.         "Geometrie.wkt": {
72.           "@value": "<http://www.opengis.net/def/crs/EPSG/0/4326> LINESTRING (3.24239104986191
51.2259355430829, 3.24177950620651, 51.2256331896095, 3.24177950620651  51.2256331896095)",
73.           "@type": "geosparql:wktLiteral"
74.         }
75.       }
76.     },
77.     {
78.       "@id": "_:wgkn001",
79.       "@type": "Wegknoop",
80.       "Wegknoop.geometrie": {
81.         "@type": "Punt",
82.         "Geometrie.gml": {
83.           "@value": "<gml:Point
srsName=\"http:\\//www.opengis.net/def/crs/EPSG/0/4326\"><gml:coordinates> 3.24239104986191,
51.2259355430829</gml:coordinates><gml:Point>",
84.           "@type": "geosparql:gmlLiteral"
85.         }
86.       }
87.     },
88.     {
89.       "@id": "_:wgkn002",
90.       "@type": "Wegknoop",
91.       "Wegknoop.geometrie": {
92.         "@type": "Punt",
93.         "Geometrie.gml": {
94.           "@value": "<gml:Point
srsName=\"http:\\//www.opengis.net/def/crs/EPSG/0/4326\"><gml:coordinates>3.24177950620651,
51.2256331896095</gml:coordinates><gml:Point>",
95.           "@type": "geosparql:gmlLiteral"
```

```
 96.           }
 97.         }
 98.       },
 99.       {
100.         "@id": "_:vkmauto001",
101.         "@type": "Verkeerstelling",
102.         "Verkeerstelling.geobserveerdKenmerk": {
103.           "@type": "Verkeerstellingkenmerk",
104.           "Verkeerstellingkenmerk.kenmerktype": "cl-vkt:aantal",
105.           "Verkeerskenmerk.voertuigType": "cl-vrt:fiets"
106.         },
107.         "Verkeersmeting.geobserveerdObject": "_:mpt001",
108.         "Verkeersmeting.fenomeenTijd": ":_fenomtime001",
109.         "Verkeerstelling.tellingresultaat": 5,
110.         "Verkeersmeting.uitgevoerdMet": "_:mti001",
111.         "dct:memberOf": "_:GM001"
112.       },
113.       {
114.         "@id": "_:fenomtime001",
115.         "Verkeersmeting.fenomeenTijd": {
116.           "@type": "time:ProperInterval",
117.           "time:hasBeginning": {
118.             "@type": "time:Instant",
119.             "time:inXSDDateTime": {
120.               "@type": "xml-schema:dateTime",
121.               "@value": "20161122T09:00:00.000Z"
122.             }
123.           },
124.           "time:hasEnd": {
125.             "@type": "time:Instant",
126.             "time:inXSDDateTime": {
127.               "@type": "xml-schema:dateTime",
128.               "@value": "20161122T10:00:00.000Z"
129.             }
130.           }
131.         }
132.       },
133.       {
134.         "@id": "_:mpt001",
135.         "@type": "Verkeersmeetpunt",
136.         "Verkeersmeetpunt.geometrie": {
137.           "@type": "Punt",
138.           "Geometrie.gml": {
139.             "@value": "<gml:Point
srsName=\"http:\\//www.opengis.net/def/crs/EPSG/0/4326\"><gml:coordinates>3.24177950620651,
51.2256331896095</gml:coordinates><gml:Point>",
140.             "@type": "geosparql:gmlLiteral"
141.           }
142.         },
143.         "Verkeersmeetpunt.verkeersmeetpuntnetwerkreferentie":"_:pr001",
144.         "Verkeersbemonsteringsobject.bemonsterdObject": "_:rri001"
145.       },
146.       {
147.           "@id": "_:pr001",
148.           "@type": "Puntreferentie",
149.           "Puntreferentie.opPositie": {
150.             "@type": "Lengte",
151.             "KwantitatieveWaarde.waarde": "300",
152.             "KwantitatieveWaarde.standaardEenheid": {
153.               "@value": "m",
```

```
154.             "@type": "ucum:ucumunit"
155.           }
156.         },
157.         "Linkreferentie.toepassingsRichting": "cl-trt:bothDirections"
158.       },
159.     {
160.       "@id": "_:mti001",
161.       "@type": "Sensor",
162.       "Systeem.type": "cl-mit:rubberslang"
163.     }
164.   ]
165. }
166.
```

Lines 1 to 31 contain the context, i.e. the labels replaced in the text by persistent URIs at machine level. Lines 34 to 54 contain the general description of the dataset.

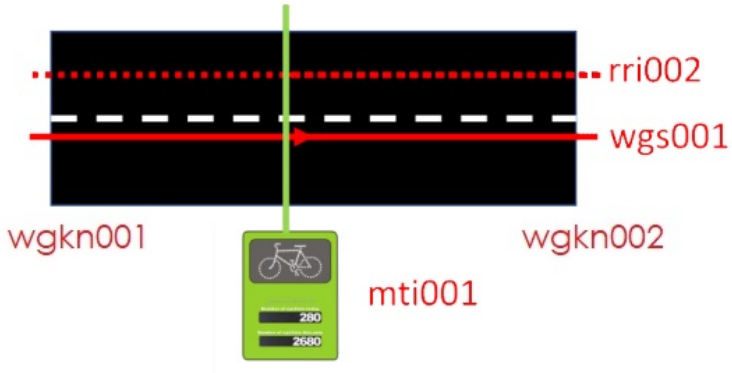
From line 66 to line 74, the road segment is defined. An object named "wgs001" is created, which is of type "road segment" and has a start node ("wgkn001") and end node ("wgkn002"). Then the geometry is described as a line string, with three shape points, namely the start point of the road segment, the measurement point and the end point of the road segment. In this way, the data always contains a road segment on which the measurement point lies. If more shape points are included in the input data for the road segment, these can be included in the linestring for a more accurate shape.

Then the two road nodes are described, with the type "road node" and a geometry as a point according to GML syntax. During the transformation from JSON to JSON-LD, the GeoJSON notation is converted to GML to be OSLO compatible.

It contains two more objects of interest for the location, namely the direction of travel and a point reference. The point reference contains as value the offset of the measurement point from the starting point of the road segment (expressed in metres, here 300 metres), with specification of the unit used ("m").

The direction of travel is an object ("rri001") referring to the road segment wgs001, which has as application direction "inDirection", i.e. from A node to B node. This object is used as a reference when defining the measurement at this location, this will be explained in a moment.

This figure (see also earlier) illustrates this :



The input data for the measurement is as follows :

```
 1. {
 2.   "id": "5ff87dcd9dc02be13c259e28",
 3.   "type": "atc",
 4.   "origin": "A",
 5.   "destination": "B",
 6.   "classification": "fiets",
 7.   "timestamp": "2016-11-22T09:00:00.000Z",
 8.   "count": 5,
 9.   "poiId": "05d50e16-2e4e-414a-921a-91a1ea11cb02",
10.   "surveyId": "b18222b9-7cd9-4c3c-adbe-41c0c597ee7d"
11. }
12.
```

Again, the data starts with an ID, a field "type" which refers to sensor used, a reference to start point and end point of the measured road section ("origin" and "destination"), a vehicle type (bicycle), a timestamp (date and time), the measurement result, and an ID for the location and an ID for the counting campaign.

Looking at the resulting OSLO-compliant JSON-LD format (above), we see this. On line 100, an object with the ID "vkmauto001" is defined. This gets as type "traffic count", of type "number" (in other words, a count) of vehicles of type "bicycle" (line 105).

Line 107 states that the measurement is made at the measurement point defined above (in semantic terms : the observed object of the measurement is the measurement point). Line 108 states that the object "phenomtime001" contains the time slot (see below), line 109 contains the counting result, and line 110 states that the measurement was performed with object mti001, which is further defined. Finally, the count is considered a member of dataset GM001.

From line 115, the time period is defined (object "phenomtime001"), as an interval with a beginning (from line 117) and an end (from line 124).

From line 160, the measuring instrument is defined (object "mti001") as a sensor of type "rubber tube".

This example shows how data transforms from a non-linked data file in JSON to linked data according to the OSLO standard in JSON-LD format. This is for illustrative purposes, the IT components described further immediately translate this data into database objects to be offered to the LDES server.

2.5 ARCHITECTURE

Digital Flanders provides several building blocks to unlock traffic data via VSDS as open data. In short, these building blocks do two things: convert data to the relevant OSLO model (in this case, the implementation model Cross-section Counting), and unlock data via an LDES server to the outside world.

Technical documentation on these components can be found here : https://informatievlaanderen.github.io/VSDS-Tech-Docs/introduction/LDES_server. The server components themselves can be found here : https://hub.docker.com/r/ldes/ldes-server/tags, and the code is publicly available on GitHub : https://github.com/informatievlaanderen/VSDS-LDESServer4j.

There are two possible architectures that can be used, namely a data stream processed via APACHE NIFI components and offered to the server, or a data stream that uses the LDIO (Linked Data Interaction Orchestrator) components for this purpose. Apache NiFi (https://nifi.apache.org) is an open source software

package that has a very comprehensive feature set for all kinds of data processing and visualisation purposes. LDIO (https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/) are Java components developed specifically for the Flemish Smart Data Space (VSDS).

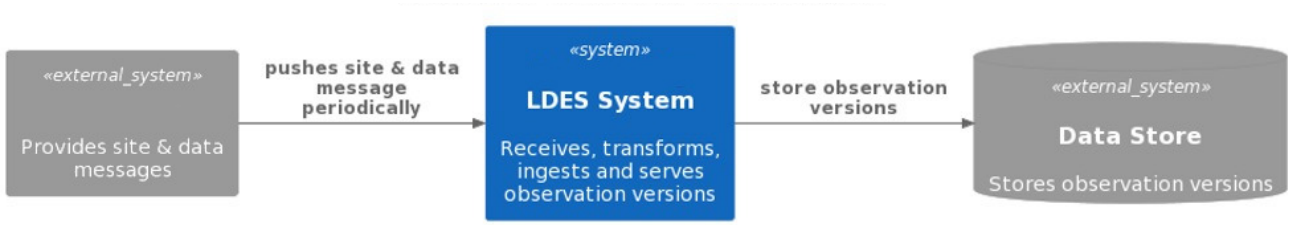
Because of the complexity of Apache NiFi, it is chosen here to work with LDIO components. For very large and complex datasets, Apache NiFi may be a better choice because of robustness and features, but for 90% of applications, LDIO is the better choice, hence the description is further based on this.

The components consist of 5 categories :

- Input
- Core
- Output
- Transformers
- Adapters

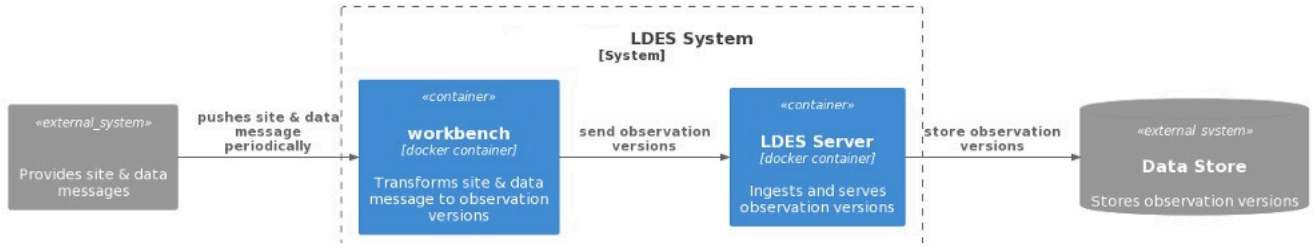
The input components read data either directly from a database or via an API (push or pull). The adapter components contain functionality to go from regular data (e.g. JSON or NGSI) to linked data. The transformers contain functionality to edit the data content, such as a function to convert GeoJSON to WKT, or a query to map linked data to the OSLO model.

If we take a quick look at a real-life example, this summarises what the implementation for open data of the city of Bruges looks like.

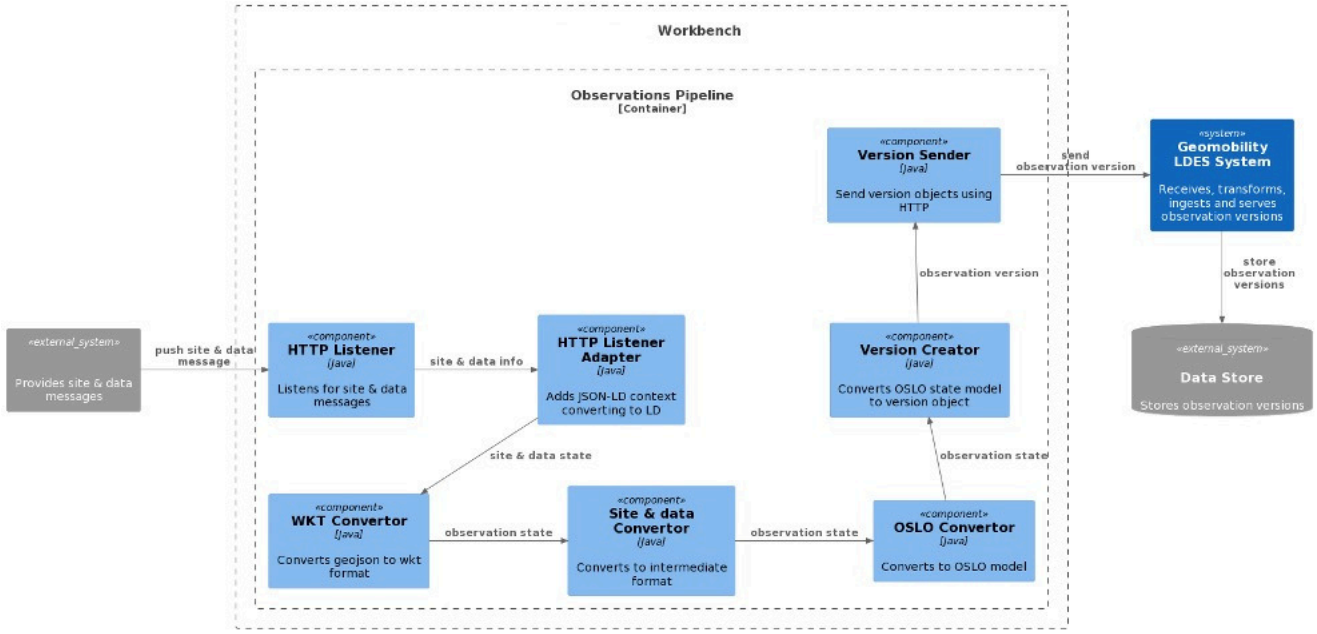


From the Backend system, files are pushed that contain (in this case in JSON format) the locations and counts. The LDES system takes this data, processes it, and offers it to the outside world.

If we break this down functionally further, we arrive at the following figure, where the LDES system breaks down into 2 major parts : the workbench, where the data input and conversion happens, and the LDES server that offers the data to the outside world.



If we also break down the workbench further, we arrive at the following components :



- - HTTP listener : In the example, data is pushed from the backend system, the HTTP listener picks up this data and passes it to the next component. https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ldio/ldio-inputs/ldio-http-in
- - HTTP listener adapter : converts JSON to JSON-LD by adding a context. https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ldio/ldio-adapters/ldio-json-to-json-ld
- - WKT converter : this converts coordinates in geojson format to Well known text (WKT) https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ldio/ldio-transformers/ldio-geojson-to-wkt
- - Site and data converter : This combines site data and measurement data into an intermediate linked data format. This is not yet OSLO compliant. https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ldio/ldio-transformers/ldio-sparql-construct
- - OSLO converter : This is where the data is made OSLO compliant. This and the previous component use a SPARQL construct query as input. https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ldio/ldio-transformers/ldio-sparql-construct
- - Version creator : creates versioned objects from the data (immutable objects, see LDES specs) https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ldio/ldio-transformers/ldio-version-object-creator
- - Version sender : sends to LDES server https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ldio/ldio-outputs/ldio-http-out

This is just an example of a workflow, but it contains the key components. In summary, the following steps are necessary :

- Bringing in data (either with an HTTP listener or an HTTP poller/push vs. pull)

- Transforming data to Linked data, possibly combining location data and measurement data
- Perform any transformations (e.g. convert/calculate coordinates)
- Convert data to the OSLO model
- Convert data to immutable objects
- Send data to the LDES Server.

All documentation regarding the configuration of the LDIO components and the LDES Server can be found here : https://informatievlaanderen.github.io/VSDS-Tech-Docs/.

The latest version (v2.5) can be found here : https://hub.docker.com/r/ldes/ldes-server/tags.

The latest version (v1.11.0) of the LDIO components can be found here : https://hub.docker.com/r/ldes/ldi-orchestrator/tags.

A quick guide on setting up an LDES server (using GIPOD data as an example) can be found here : https://informatievlaanderen.github.io/VSDS-Tech-Docs/quickstart/Publish_LDES.

2.6 IMPLEMENTATION PROCES

The implementation process to disclose cross-sectional census data as OSLO compliant open data via LDES includes the following steps :

- - Preparation
- - OSLO mapping
- - Configuring LDIO
- - Deployment
- - Metadata deployment

2.6.1 Preparation

In the preparation phase it is checked whether the existing data contains all the information that is necessary (as described above). If this data is not present, it must be checked whether the data can be enriched (once in the case of a historical dataset, permanently in the case of an ongoing census). A typical example is the description of a location, where the model expects a road segment with offset. If that data is missing (e.g., because only an XY coordinate is present in the data), it should be checked whether the data can be added within the existing backend system (e.g., by permanently importing data into the source database), or a preprocessing script can be worked out that retrieves and combines data before submitting it to the LDIO components (e.g., with a python script that retrieves road segments from the GRB).

Second, the hosting of the LDIO components must be decided. These run in a docker container that can be hosted on a server. Depending on the backend architecture, one can choose to run these docker containers on a server next to the source database (easier for data access), on a remote server (with API access to the data) or in the cloud (AWS, Azure...). Depending on the size of the dataset (number of records), storage size should be determined. For cross-sectional counts, currently 2.5KB per record can be counted as a rule of thumb.

2.6.2 OSLO mapping

Secondly, a mapping is worked out from the current data to the OSLO model. Where the OSLO model uses a code list (e.g. for vehicle category, speed measurement type and sensor type) a table needs to be created that maps the values in the source data to the desired value in the OSLO model. Some expected values in the OSLO model will simply not be present in the source data because they are a constant. For example, it is common that all measurements in a data set were made with the same type of sensor, and that type is not explicitly in the data. The same reasoning often applies to driving direction : usually all counts of a certain type have been performed in the same direction. This should also be documented.

2.6.3 Configuration LDIO

Next, the LDIO components can be configured. This involves the following :

- - Initialization of docker containers via YML. A total of 4 containers are initialized : LDIO workbench, LDES Server, MongoDB, NGINX (reverse proxy). The last two components can of course be exchanged for a database and reverse proxy of your choice.
- - Configuration of the LDIO components. This is done by modifying a YML configuration file. Some components (transformation to linked data and transformation to OSLO) require the creation of a SPARQL query (.RQ format) to be included in the configuration. The sparql construct query builds the data according to the written query and writes it away as linked data.

2.6.4 Deployment

After configuration (and testing on local docker containers), deployment can proceed. This involves spinning up the necessary docker containers in the correct locations with correct configuration files. Then the data can be pushed to the configuration if applicable.

2.6.5 Metadata deployment

To make data discoverable, it is necessary to send the metadata (dcat) of the dataset to the server using linked data, in pieces relevant to the LDES stream in question and the defined views. This is done by modifying and forwarding (seeding) a TTL file. The server glues these pieces together and creates a DCAT catalog, which can be picked up by a metadata crawler (such as data finder).

2.7 REFERENTIES

https://data.vlaanderen.be/standaarden/ : standards register

https://data.vlaanderen.be/standaarden/standaard-in-ontwikkeling/applicatieprofiel-verkeersmetingen.html : Applicatieprofile Verkeersmetingen

https://data.vlaanderen.be/standaarden/standaard-in-ontwikkeling/vocabularium-verkeersmetingen.html : vocabularium verkeersmetingen

https://informatievlaanderen.github.io/VSDS-Tech-Docs/introduction/LDES_server : LDES Server documentation

https://hub.docker.com/r/ldes/ldes-server/tags : LDES Server docker images

https://informatievlaanderen.github.io/VSDS-Tech-Docs/introduction/LDES_client : LDES client technical documentation

https://github.com/Informatievlaanderen/VSDS-Linked-Data-Interactions : LDIO componenten repository

https://implementatie.data.test-vlaanderen.be/doc/implementatiemodel/verkeersmetingen/ : implementationmodel sectional counts

http://www.openlr.org : openLR organisation

https://github.com/tomtom-international/openlr : Open LR Repository

http://www.openlr.org/fileadmin/user_upload/openlr-whitepaper_v1.5.pdf : openLR specifications

https://implementatie.data.test-vlaanderen.be/doc/implementatiemodel/verkeersmetingen/ontwerpstandaard/2023-09-27/context/Verkeersmetingen-im.jsonld : Sectional counts JSONLD context

https://nifi.apache.org : apache NiFi

https://informatievlaanderen.github.io/VSDS-Linked-Data-Interactions/ : LDIO project

https://informatievlaanderen.github.io/VSDS-Tech-Docs/quickstart/Publish_LDES : quick guide LDES client