

OpenDataHub Docs Documentation

Release 2022.03

The Open Data Hub Team

Jun 30, 2023

TABLE OF CONTENTS

1	Quickstart	1
2	Introduction	3
2.1	Project Overview	3
2.2	Getting Involved	4
2.2.1	How to Collaborate	5
2.2.2	Bug Reporting and Feature Requests	7
2.3	Accessing the Open Data Hub	7
2.3.1	Browser access	7
2.3.2	Programmatic access	8
2.3.3	The Open Data Hub Virtual Knowledge Graph	9
2.3.4	The AlpineBits Client	14
3	Domains and Datasets	15
3.1	Data Providers	16
3.2	Datasets, Open Data, and Licenses	17
3.2.1	Authentication	17
3.3	Datasets in the Mobility Domain	17
3.3.1	Technical Information for Mobility Dataset	17
3.3.2	Traffic	27
3.3.3	Mobility	30
3.4	Datasets in the Tourism Domain	32
3.4.1	Technical Information for Tourism Dataset	32
3.5	Datasets in Other Domains	41
4	List of HOWTOs	43
4.1	Mobility	43
4.1.1	How to Access Mobility Data With API v2	43
4.1.2	How to Access Analytics Data in the Mobility Domain	45
4.2	Tourism	49
4.2.1	How to access Tourism Data?	49
4.2.2	How to use the Open Data Hub's Tourism Data Browser?	51
4.2.3	Quick and (not-so) Dirty Tips for Tourism (AKA Mini-howtos)	54
4.2.4	How to access Open Data Hub AlpineBits Server as a client	57
4.3	Generic HOWTOs	59
4.3.1	How to use authentication?	59
4.3.2	How to set up your local Development Environment?	62
4.3.3	How to set up Postman (API Development Environment)?	64
4.3.4	How to insert and modify NOI Events?	68
4.3.5	How to Publish your Web Component on the Open Data Hub Store	71

4.3.6	How to Access Open Data Hub Data Using SPARQL	71
4.3.7	How to Access Open Data Hub Data With R and SPARQL	73
5	Resources for Developers	77
5.1	Guidelines for Developers	77
5.1.1	Platform Guidelines - Bignami Version	77
5.1.2	Database Guidelines - Bignami Version	78
5.2	Platform Guidelines - Full Version	79
5.2.1	Programing Languages, Environments, and Related Technologies	79
5.2.2	Platforms and Architectural Considerations	83
5.3	Database Guidelines - Full Version	85
5.3.1	Database design methodology	86
5.3.2	Stored procedures and functions, foreign data wrappers	88
5.3.3	Indices and Partioning	88
5.3.4	Encoding, collation and localization	88
5.4	Development, Testing, and Production Environments	90
5.5	Authentication in the Open Data Hub	91
5.5.1	Keycloak for API v2	91
5.5.2	OAuth2 Authentication	91
5.6	Authentication To Internal Infrastructure	92
5.6.1	Quick howto	92
5.7	GITHUB Quick Documentation for Contributors	94
5.7.1	Prerequisites	94
5.7.2	Project Checkout	94
5.7.3	Create a pull request	96
5.7.4	Syncing a Fork	97
5.7.5	Resolving Merge Conflicts	98
6	Apps Built From Open Data Hub Datasets	101
6.1	Production Stage Apps	101
6.2	Beta Stage Apps	102
6.3	Alpha Stage Apps	102
7	Appendices	105
7.1	Open Data Hub Architecture	105
7.1.1	The Elements of the Open Data Hub in Details	106
7.2	Glossary	107
7.3	Licenses and TOS for the Open Data Hub material	109
7.3.1	The FLOSS four freedoms	109
7.3.2	Licenses for Open Data Hub resources	109
7.3.3	APIs Terms of Service	111
7.4	Changelog	111
7.4.1	2021.03	112
7.4.2	2021.04	112
7.4.3	2021.05	113
7.4.4	2021.06	113
7.4.5	2021.08	114
7.4.6	2021.09	114
7.4.7	2021.12	114
7.4.8	2022.03	115
7.4.9	2022.03	115
8	Frequently Asked Questions	117
	Index	119

QUICKSTART

New in version 2022.03: Quickstart section

This section helps you in getting quickly in touch with the most popular tools developed by Open Data Hub to interact with the dataset and the whole ecosystem.

At the time of writing, all tools that have been developed are available online: they are summarised in the following table.

Data Browser

The [ODH Tourism Data Browser](#) allows to access all the Open Data available in the Tourism domain by simply browsing the content of the various datasets.

You can simply use those data for your convenience, or you might even find a novel way to exploit those data and use them in an app or portal you are going to develop. A detailed howto is available: [How to use the Open Data Hub's Tourism Data Browser?](#) to help you getting acquainted with the browser.

Currently, the Data Browser is being developed to be able to access also data in the Mobility domain as well.

Analytics tools

Open the **Analytics for Mobility** web page, at <https://analytics.opendatahub.bz.it/> This portal uses data in the mobility domain to display various information about the sensors, including their locations, what they measure, and actual data in near-real time. You can retrieve data gathered by the sensors directly from the dataset, in almost real-time.

Swagger API interface

The Open Data Hub team has developed REST APIs to programmatically access both the Tourism and Mobility domains. You can access their swagger interface to browse the data and learn how to use the API. You will be guided in building the query and will receive as a result both all the data satisfying the query and the full query in a format that you can use with the **curl** command from a shell or a script.

Mobility resources

- The URL of the Swagger interface is <https://mobility.api.opendatahub.bz.it/>
- Check out the *Mobility dedicated howto* ([How to Access Mobility Data With API v2](#)) that will guide you in the first steps.

Tourism resources

- The URL of the Swagger interface is <https://tourism.opendatahub.bz.it/swagger/ui/index>
- Check out the *Tourism dedicated howto* (*How to access Tourism Data?*) that will guide you in the first steps.

Web Component Store

Web Components are a W3C's effort to provide standard components for the Web. Anyone can create a Web Component, using standard languages like CSS, JSON, HTML and share it with other, in this case with the Open Data Hub community.

If you have developed a *Web Component* that you deem useful for the Open Data Hub project or that can be used on top of data provided by the Open Data Hub, you can share it and allow others to reuse it, by making it freely available on Open Data Hub's [Web Components Store](#).

The only requirement for all the Web Components offered through the Store is that they **must** be released as an *Open Source Licence*, compatible with those used within the Open Data Hub project.

To help you in the process of publishing your Web Component in Open Data Hub's store, check the howto: *How to Publish your Web Component on the Open Data Hub Store*.

See also:

More information about the Open Data Hub project, its goal, and possibility to interact or collaborate with it can be found in sections *Project Overview*, *Getting Involved*, *Open Data Hub Architecture*, *Domains and Datasets*, *Accessing the Open Data Hub*.

INTRODUCTION

Changed in version 2022.03: moved sections *Getting Involved* and *Accessing the Open Data Hub* as part of this section.

This is the website of the Open Data Hub documentation, a collection of technical resources about the Open Data Hub project. The website serves as the main resource portal for everyone interested in accessing the data or deploying apps based on *datasets* & *APIs* provided by the Open Data Hub team.

The technical stuff is composed of:

- Catalogue of available datasets.
- How-tos, FAQs, and various tips and tricks for users.
- Links to the full API documentation.
- Resources for developers.

For non-technical information about the Open Data Hub project, please point your browser to <https://opendatahub.bz.it/>.

2.1 Project Overview

The Open Data Hub project envisions the development and set up of a portal whose primary purpose is to offer a single access point to all (Open) Data from the region of South Tyrol, Italy, that are relevant for the economy sector and its actors.

The availability of Open Data from a single source will allow everybody to utilise the Data in several ways:

- Digital communication channels. Data are retrieved from the Open Data Hub and used to provide informative services, like newsletters containing weather forecasting, or used in hotels to promote events taking place in the surroundings, along with additional information like seat availability, description, how to access each event, and so on and so forth.
- Applications for any devices, built on top of the data, that can be either a PoC (Proof of Concept) to explore new means or new fields in which to use Open Data Hub data, or novel and innovative services or software products built on top of the data.
- Internet portals and websites. Data are retrieved from the Open Data Hub and visualised within graphical charts, graphs, or maps.

There are many services and software that rely on Open Data Hub's Data, which are listed in the *Apps Built From Open Data Hub Datasets* section, grouped according to their maturity: production stage, beta and alpha stage.

Figure 2.1 gives a high level overview of the flow of data within the Open Data Hub: at the bottom, *sensors* gather data from various domains, which are fed to the Open Data Hub Big Data infrastructure and made available through endpoints to (third-party) applications, web sites, and vocal assistants. A more technical and in-depth overview can be found in Section *Open Data Hub Architecture*.

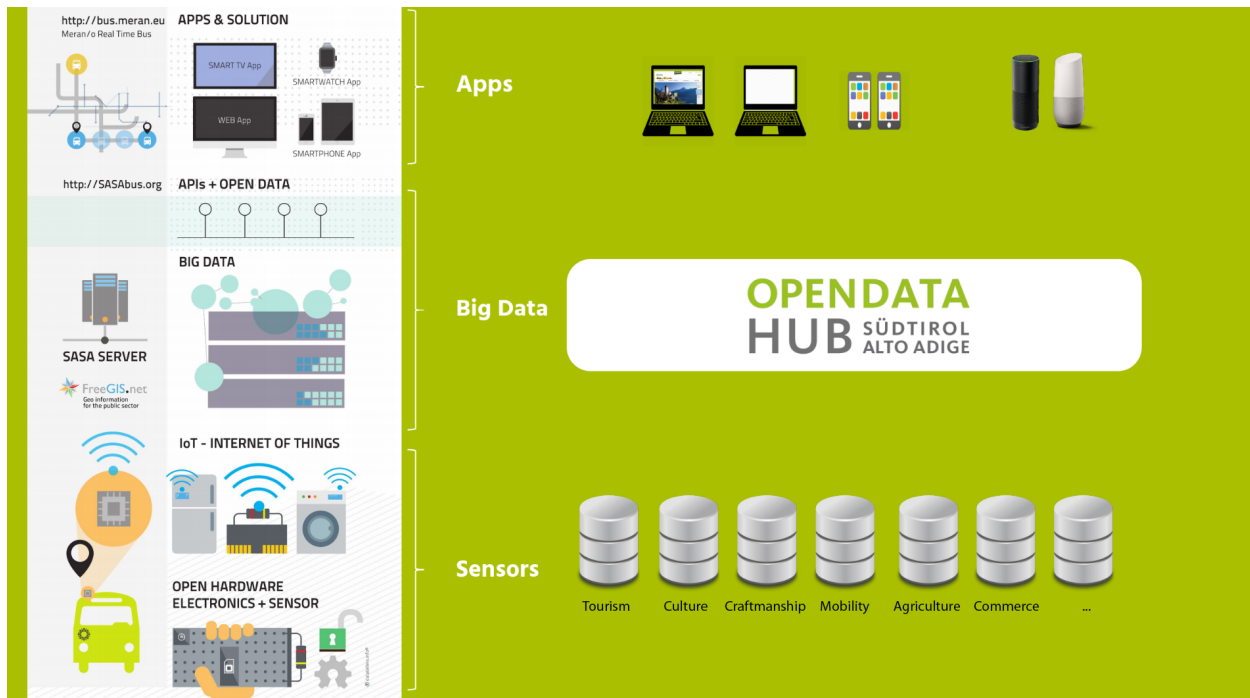


Figure 2.1: An overview of the Open Data Hub Project.

All the data within the Open Data Hub will be easily accessible, preferring open interfaces and APIs which are built on existing standards like [The Open Travel Alliance](#) (OTA), [The General Transit Feed Specification](#) (GTFS), [Alpinebits](#).

The Open Data Hub team also strives to keep all data regularly updated, and use standard exchange formats for them like [Json](#) and the [Data Catalog Vocabulary](#) (DCAT) to facilitate their spreading and use. Depending on the development of the project and the interest of users, more standards and data formats might be supported in the future.

2.2 Getting Involved

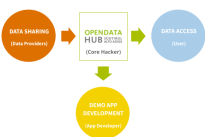


Figure 2.2: Relations between the actors involved in the Open Data Hub.

The Open Data Hub team welcomes new partners and user to join the project. For this reason, a number of services are offered to help you getting in touch with the project and support you in case you want to either collaborate with the project or simply use the data made available.

The Open Data Hub team envisioned different possibilities to join the project, that are detailed in the remainder of this section, including—but not limited to—reporting bugs in the API or errors in the API output, making feature requests or suggestions for improvement, and even to participate in the development or supply new data to be shared through the Open Data Hub.

[Figure 2.2](#) shows at a glance the services offered by the Open Data Hub together with the various roles that potential users can play within the Open Data Hub, according to their interest, expertise, skills, and knowledge.

Data Sharing

The Open Data Hub team helps you in various activities: give **visibility** to your data, identify both **suitable, common formats**, processing algorithms and licenses to share your data, and to use popular technologies known by application developers.

This service is suitable for [Data Providers](#).

Data Access

The Open Data Hub team supports software development companies to **access to real-time data**, by making available to everyone a stable **communication channel** which uses a **machine-readable** protocol and an **Open Data license**, to ensure everybody can easily find all the available data.

This service is most suitable for *Data Consumers* and App developers.

Demo App Development

The Open Data Hub team collaborates with companies to design and develop Proof of Concepts and demo software, released under an *Open Source License* that use data offered by the Open Data Hub. All the software can then either be used as-is—to show and test its potentials, or as a guideline and inspiration to develop new digital products.

This service is suitable for *Data Consumers*, App developers, and companies interested in development built on top of the Open Data Hub.

Besides the tasks that you find below, you can also help the Open Data Hub project grow and improve by reporting bugs or asking for new features, by following the directions presented in *the dedicated section below*.

2.2.1 How to Collaborate

This section gives an overview of which tasks you can play when collaborating with the Open Data Hub project.

Depending on your interest on the Open Data Hub Project, we welcome your participation to the project in one of the roles that we have envisioned: **User (Data Consumer)**, **App developer**, **Core Hacker**, and **Data Provider**. Below you can find out a list of tasks that belong to each of these roles; we believe that this list can help you understand which type of contribution you can give to the Open Data Hub project!

As a **User** I can...

...install and use an app built on top of the API.

Browse the list of *available applications developed by third-parties* that use Open Data Hub data, choose one that you are interested in, install it and try it out, then send feedback to their developers if you feel something is wrong or missing.



...explore the data in the datasets.

Choose a dataset from the list of *Domains and Datasets* and start gathering data from it, by using the documentation provided in this site. You can then provide any kind of feedback on the dataset: reports about any malfunctions, suggestions for improvements or new features, and so on.

Moreover, if you are interested in datasets that are not yet in our collection, get in touch with the Open Data Hub team to discuss your request.

As a **Data Provider** I can...

...provide Open Data to the Open Data Hub project.

Share with an Open Data Licence (like e.g.,  or ) the data you own, that can prove interesting for the Open Data Hub, for example because they complement existing data in the Open Data Hub or they pertain to an area which is not yet covered. Let your Open Data be freely used by App Developers in their applications.

Note: A *Data Provider* is an entity (be it a private company, a public institution, or a citizen) that gathers data on a regular basis from various sensors or devices and stores them in some kind of machine-readable format.

As an **App Developer** I can...

...harvest data exposed by the dataset.

Browse the list of [Domains and Datasets](#) to see what types of data are contained in the datasets, and think how they can be used.

For this purpose, we maintain an updated list of the [available datasets](#) with links to the API to access them.

...build an application with the data.

Write code for an app that combines the data you can harvest from the available datasets in various, novel way.

To reach this goal, you need to access the APIs, their documentation, and the datasets. It is then your task to discover how you can reuse the data in your code.

...integrate Open Data Hub data using Web Components.

The Open Data Hub team and their partner have developed a [small library of Web Components](#) that can be integrated in existing web sites or used as guidance to develop new Web Components.

...publish my app in Open Data Hub.

As soon as you have developed a stable version of your app, get in touch with us: We plan to maintain an updated list of apps based on our dataset included with this documentation.

No software installation is needed: Go to the list of [available applications developed by third-parties](#) that use Open Data Hub data, to the API documentation of each [Dataset](#) and start from there, and develop in a language of your choice an application that uses our data.

As a **Open Data Hub Core Hacker** I can...

...help shape the future of Open Data Hub.

Participate in the development of Open Data Hub: Build new data collectors, extend the functionality of the broker, integrate new datasets on the existing infrastructure, develop new stable API versions.

To be able to become a core hacker, however, requires a few additional tasks to be carried out:

1. Learn how to successfully integrate your code with the existing code-base and how to interact with the Open Data Hub team. In other words, you need to read and accept the [Guidelines for Developers](#) (click on the link for a summary), which are available in two extended, separate parts: [Platform Guidelines - Full Version](#) and [Database Guidelines - Full Version](#).
2. Understand the [Open Data Hub Architecture](#).
3. Learn about the [Development, Testing, and Production Environments](#).
4. Install the necessary software on your local workstation (be it a physical workstation, a virtual machine, or a Docker instance), including PostgreSQL with postgis extension, JDK, git.
5. Set up all the services needed (database, application server, and so on).
6. Clone our git repositories. To successfully complete these tasks, please read the [How to set up your local Development Environment?](#) tutorial, which guides you stepwise through all the required set up and configuration, along with some troubleshooting advice.
7. Coding. That's the funniest part, enjoy!

To support the installation tasks and ease the set up of your workstation, we are developing a script the you will do the job for you. Stay tuned for updates.

2.2.2 Bug Reporting and Feature Requests

This section explains what to do in case you:

1. have found an error or a bug in the APIs;
2. like to suggest or require some enhancement for the APIs;
3. have some requests about the datasets
4. find typos or any error in this documentation repository;
5. have an idea for some specific tutorial.

If your feedback is related to the Open Data Hub Core, including technical bugs or suggestions as well as requests about datasets (i.e. points 1. to 3. above), please insert your issues on the following website:

<https://github.com/noi-techpark/bdp-core/issues>

If your feedback is related to the Open Data Hub Documentation, please insert your issue on the following website, using the template that suits your needs:

- [Report a bug](#)
- [Leave your feedback](#)

Note: You need to have a valid github account to report issues and interact with the Open Data Hub team.

We keep track of your reports in our bug trackers, where you can also follow progress and comments of the Open Data Hub team members.

2.3 Accessing the Open Data Hub

There are lots of alternatives to access the Open Data Hub and its data: interactive and non-interactive, using a browser or the command line.

Various dedicated tutorials are available in the [List of HOWTOs](#) section to help you getting started, while in section [Getting Involved](#) you can find additional ways to collaborate with the Open Data Hub Team and use the data.

2.3.1 Browser access

Accessing data in the Open Data Hub by using a browser is useful on different levels: for the casual user, who can have a look at the type and quality of data provided; for a developer, that can use the [REST API](#) implemented by the Open Data Hub or even check if the results of his app are coherent with those retrieved with the API; for everyone in order to get acquainted with the various methods to retrieve data.

Besides the online tools developed by the Open Data Hub and described in section [Quickstart](#), these other resources can be access using a browser.

Go to the [Apps Built From Open Data Hub Datasets](#) section of the documentation, particularly sub-sections [Production Stage Apps](#) and [Beta Stage Apps](#), and choose one of the web sites and portals that are listed there. Each of them uses the data gathered from one or more Open Data Hub's datasets to display a number of useful information. You can then see how data are exposed and browse them.

In the same [Apps Built From Open Data Hub Datasets](#) section, you can also check the list of the [Alpha Stage Apps](#) and choose one of them that you think you can expand, then get in touch with the authors to suggest additional features or collaborate with them to discuss its further development to improve it.

Open the [Open Data Hub Knowledge Graph Portal](#) where you can explore all the data that are already available as a virtual knowledge graph. Here you can check out some of the precooked query to see and modify them to suit your needs with the help of W3C's [SPARQL query language](#); SPARQL can be used also in the *Playground* to freely query the endpoint.

2.3.2 Programmatic access

Programmatic and non-interactive access to the Open Data Hub's dataset is possible using any of the following methods made available by the Open Data Hub team.

AlpineBits client

The AlpineBits Alliance strives to develop and to spread a standard format to exchange tourism data. Open Data Hub allows access to all data produced by AlpineBits using dedicated endpoints:

The *AlpineBits HotelData* dataset can be access from <https://alpinebits.opendatahub.bz.it/AlpineBits/>.

See also:

The dedicated howto *How to access Open Data Hub AlpineBits Server as a client*

The *AlpineBits DestinationData* endpoint is available at <https://destinationdata.alpinebits.opendatahub.bz.it/>.

Statistical Access with R

R is a free software for statistical analysis that creates also graphics from the gathered data.

The Open Data Hub Team has developed and made available *bzar*, an R package that can be used to access [BZ Analytics](#) data (see also *How to Access Analytics Data in the Mobility Domain*) and process them using all the R capabilities. Download and installation instructions, along with example usage can be found on the [bzar repository](#).

See also:

There is a howto that explains how to fetch data from the Open Data Hub datasets using R and SPARQL: *How to Access Open Data Hub Data With R and SPARQL*.

API

The APIs are composed of a few generic methods, that can be combined with many parameters to retrieve only the relevant data and then post-processed in the preferred way.

The following table summarises how the two versions of the API can be used within the Open Data Hub's domains.

API	Tourism	Mobility
v1	recommended	deprecated
v2	–	recommended

There are currently two versions of the API, v1 and v2, with the former now **deprecated** for the Mobility domain and marked as such deprecated throughout the Open Data Hub documentation. New users are recommended to use the new API v2, while users of the API v1 are encouraged to plan a migration to the new API.

The new API v2 has a different approach compared to the previous version, and therefore is not compatible with the API v1, the main difference being that all data stored in the Open Data Hub can now be retrieved *from a single endpoint*, while with API v1 there was an endpoint for each dataset.

This change in approach requires also a breaking change for the users of API v1. The initial step, indeed, will not be to open the URL of the dataset and start exploring, but to retrieve the `stationTypes` and then retrieve additional data about each station. A `stationType` is the main object of a datasets, about which all the information in a dataset relate to; a dataset includes at least one `stationType`. A new, dedicated howto describing in detail the new API v2 and a few basic examples is [already available](#) in the dedicated section of this documentation.

Note: It is important to remark that the API v2 is **only available** for datasets in the **Mobility** Domain.

CLI access

Unlike browser access, that provides an interactive access to data, with the option to incrementally refine a query, command line access proves useful for non-interactive, one-directional, and quick data retrieval in a number of scenarios, including:

- Scripting, data manipulation and interpolation, to be used in statistical analysis.
- Applications that gather data and present them to the end users.
- Automatic updates to third-parties websites or kiosk-systems like e.g., in the hall of hotels.

Command line access to the data is usually carried out with the **curl** Linux utility, which is used to retrieve information in a non-interactive way from a remote site and can be used with a variety of options and can save the contents it downloads, which can then be sent to other applications and manipulated.

The number of options required by **curl** to retrieve data from Open Data Hub's dataset is limited, usually they are not more than 3 or 4, but their syntax and content might become long and not easily readable by a human, due to the number of *filters* available. For example, to retrieve the list of all points of interests in South Tyrol, the following command should be used:

```
~$ curl -X GET "https://tourism.api.opendatahub.bz.it/v1/ODHActivityPoi?pagenumber=1&
↪pagesize=10&type=63&subtype=null&poitype=null&idlist=null&locfilter=null&
↪langfilter=null&areafilter=null&highlight=null&source=null&odhtagfilter=null&
↪odhactive=null&active=null&seed=null&latitude=null&longitude=null&radius=null" -H
↪"accept: application/json"
```

Your best opportunity to learn about the correct syntax and parameters to use is to go to the **swagger interface** of the *tourism* or *mobility* domains and execute a query: with the output, also the corresponding **curl** command used to retrieve the data will be shown.

2.3.3 The Open Data Hub Virtual Knowledge Graph

Some datasets in the Open Data Hub, namely *Accommodation*, *Gastronomy*, and *Event*, are organised into a *Virtual Knowledge Graph* that can be accessed using SPARQL from the dedicated *SPARQL endpoint*. In order to define more precise queries, this section describes the Knowledge Models (KM) underlying these datasets; the description of each KM (Knowledge Model) is accompanied by an UML diagram which shows the KM at a glance.

Besides standard W3C's OWL and RDF vocabularies, the Open Data Hub VKG uses:

- schema.org for most of the entities used
- [geosparql](https://www.geosparql.org/) for geo-references and coordinates of objects
- [Dublin Core's purl](https://www.dublincore.org/) for linking to related resources

Common Notation

Diagrams use UML class diagram formalism widely adopted in Knowledge Representation and in particular in the *W3C's Recommendation* documents for the Semantic Web. The following additional notation applies:

Prefix

The default prefix used for classes and properties is <https://schema.org/>. This means that, unless differently stated, the definition of classes and properties, including their attributes, rely on a common standard as defined in schema.org's vocabulary. As examples, see the [LodgingBusiness](#) class and the [containedInPlace](#) property.

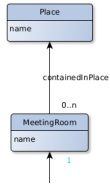
Hint: Other prefixes are explicitly pre-pended to the Class or Property name, like e.g., *noi:numberOfUnits*.

Arrows

Arrows with a white tip denote a *sub-class* relationship, while black tips denote *object properties*.

Cardinality

Cardinality of **1** is usually not shown, but implied; the **look across** notation is used. For example, the image on the right-hand side—excerpt from the *event dataset* VKG—can be read as *0 to N MeetingRooms are ContainedInPlace Place*.



Mobility Domain

The entire mobility domain has a unique underlying knowledge model, which encompasses all the datasets and therefore also allows an easier creation of cross-dataset queries. Since the mobility domain gathers data from *sensors*, useful in this domain is also the SOSA (Sensor, Observation, Sample, and Actuator) ontology, which uses **sosa** as prefix. You can check the Classes and Properties of SOSA in the [W3C's dedicated wiki page](#)

The central concept is **Station**, of which all **StationTypes** are subclass, while **Observation**, **LatestObservation**, and **ObservableProperty** are used to provide time-related information of the data gathered and relate to **Sensor**. Together with **Platform**, **Sensor** make the relation between a *Station* and its *Sensors*: For example, sensor *EChargingPlug* isHostedBy an *EChargingstation Platform*, which is also a *Station*.

The knowledge model is completed by the **Feature** superconcept, which contains also **Municipality** and **RoadSegment**, the latter defined by an *hasOriginStation* and an *hasDestinationStation*.

Accommodation Dataset

Central class in this dataset is **LodgingBusiness**, to which belong multiple **Accommodations**.

A **LodgingBusiness** has as attributes *geo:asWKT*, *email*, *name*, *telephone*, and *faxNumber* and relations

- *address* to class **PostalAddress**, which consists of *streetAddress*, *postalCode*, and *AddressLocality*
- *geo*, i.e., a geographical location, to class **GeoCoordinates**, consisting of *latitude*, *longitude*, and *elevation*

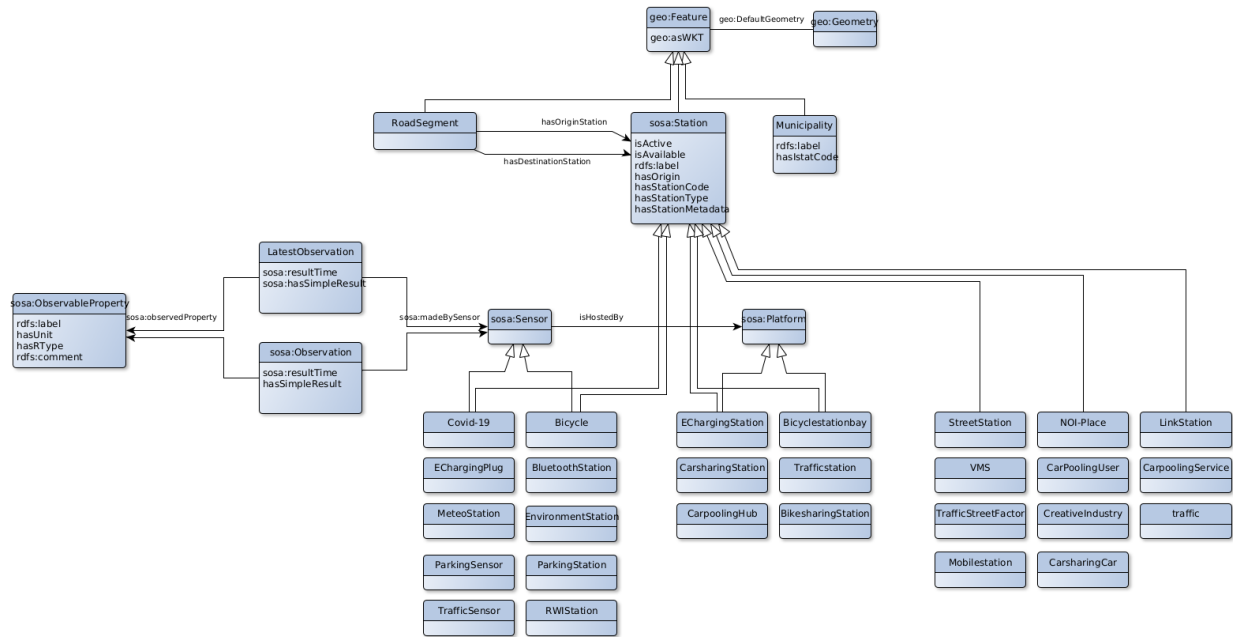
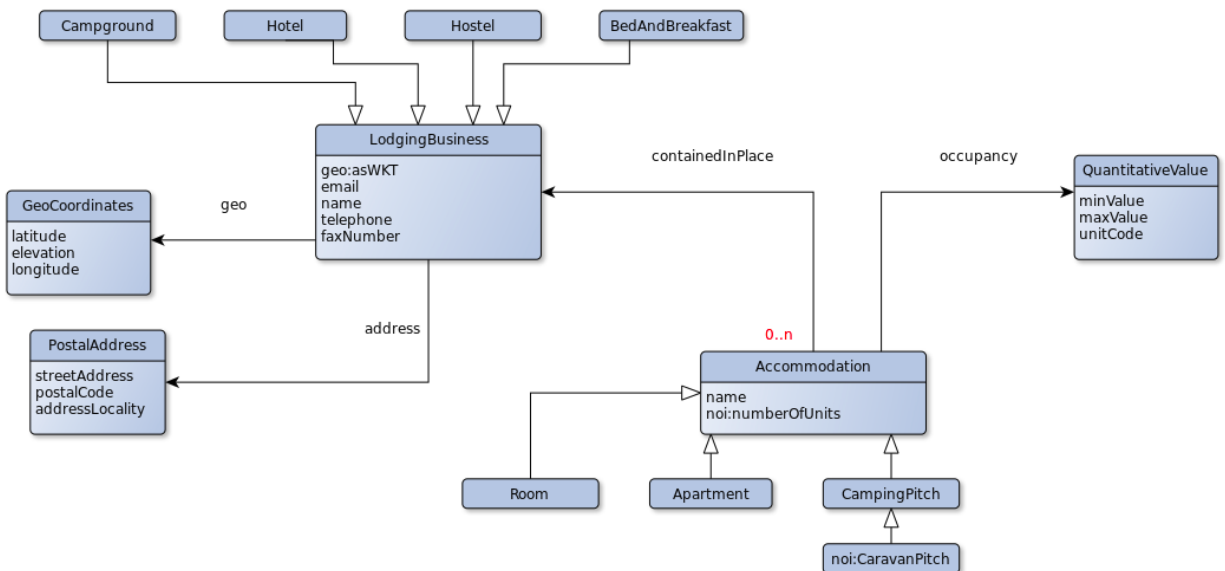
There are (sub-)types of **LodgingBusiness**—called **Campground**, **Hotel**, **Hostel**, and **BedAndBreakfast**—sharing its attributes and relations.

An **Accommodation** is identified by a *name* and a *noi:numberOfUnits* and has relations

- *containedInPlace* to **LodgingBusiness** (multiple **Accommodations** can belong to it)
- *occupancy* to **QuantitativeValue**, which gives the *maxValue* and *minValues* of available units of accommodation and a *unitCode*.

noi:numberOfUnits is the number of available rooms, suites, apartments, etc. that are available in that **Accommodation**

geo:asWKT is a method used by [opengis.net's geosparql](https://www.geosparql.org/) <<https://www.geosparql.org/>> to express geographic coordinates in a standard, textual form based on WKT (Well-known text).

Figure 2.3: The UML diagram of the *Mobility Domain*.Figure 2.4: The UML diagram of the *Accommodation Dataset*.

Gastronomy Dataset

The main class of this dataset is **FoodEstablishment**, described by *geo:asWKT*, *description*, *name*, *telephone*, and *url*.

A **FoodEstablishment** has

- a **PostalAddress**—consisting of *streetAddress*, *postalCode*, and *AddressLocality*—as *address*
- a **GeoCoordinates**—*latitude*, *longitude*, and *elevation*—as a geographical location *geo*

There are different (sub-)types of **FoodEstablishment**, all sharing the same attributes: **Restaurant**, **FastFoodRestaurant**, **BarOrPub**, **Winery**, and **IceCreamShop**.

geo:asWKT is a method used by opengis.net's *geosparql* <<https://www.geosparql.org/>> to express geographic coordinates in a standard, textual form based on WKT.

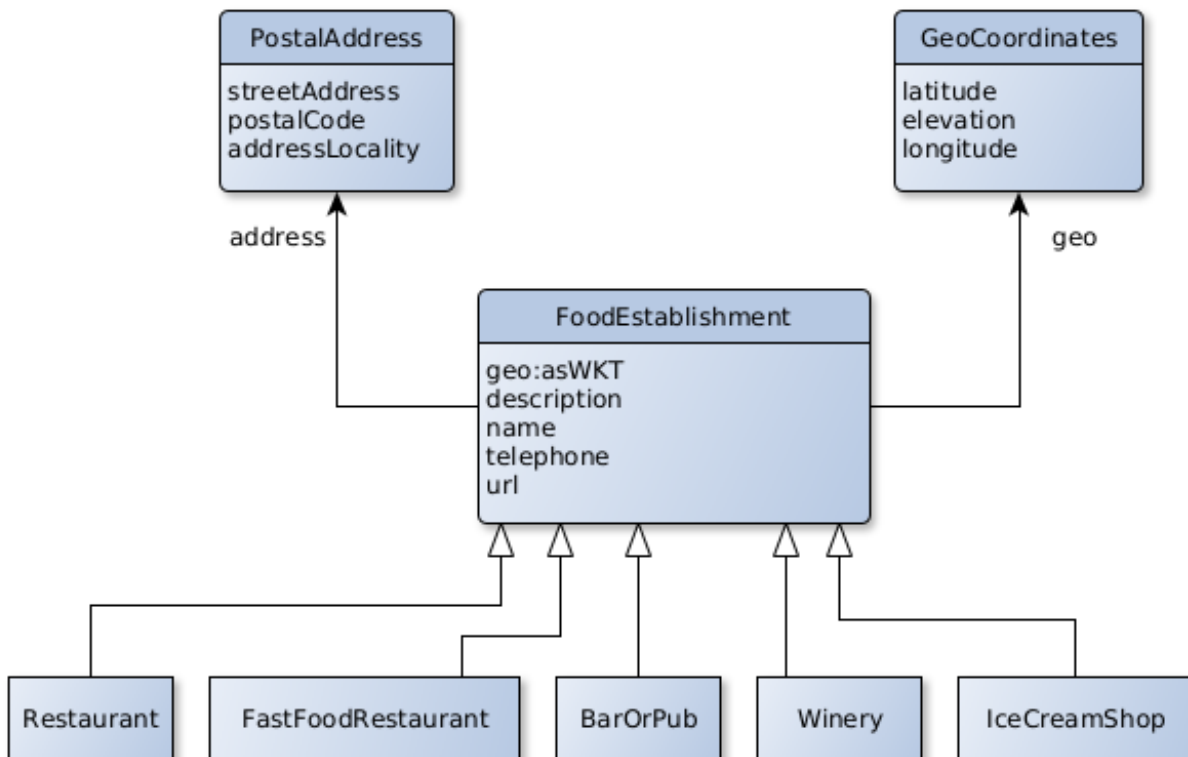


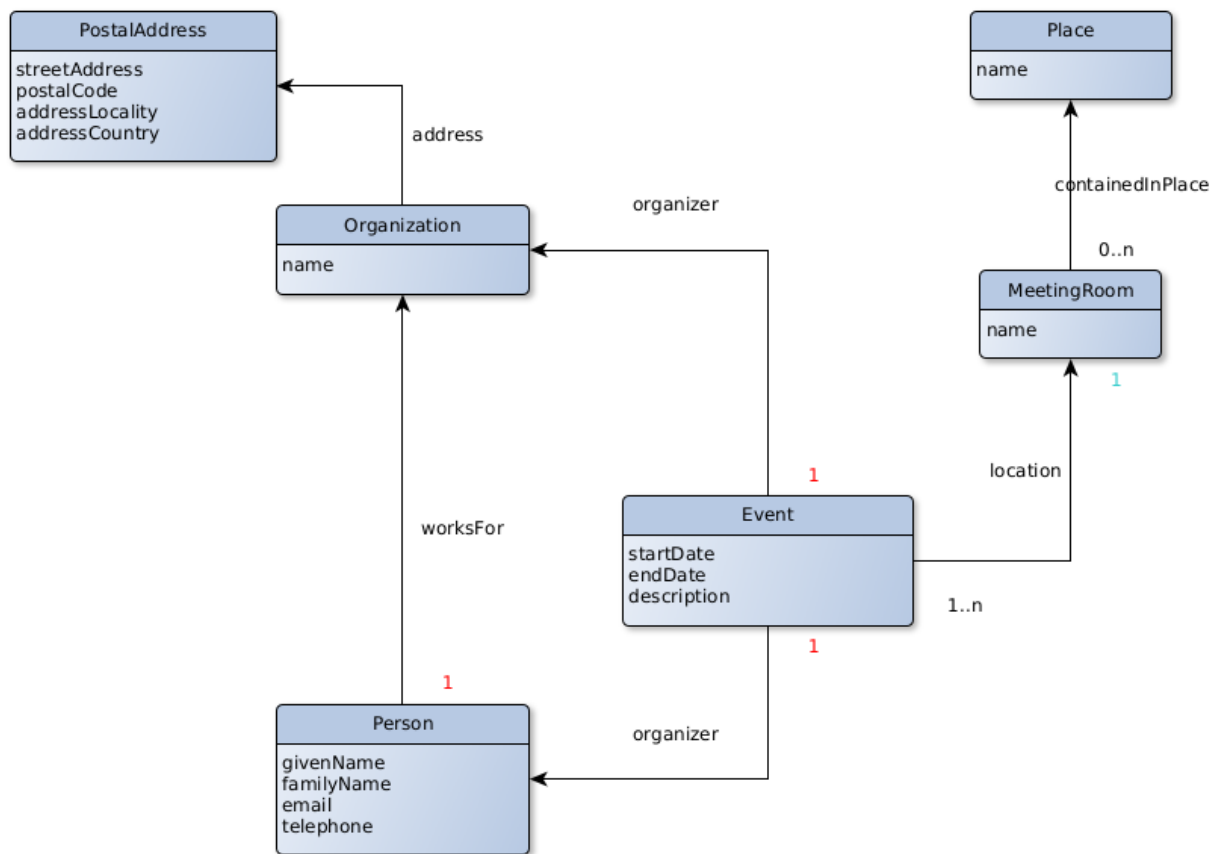
Figure 2.5: The UML diagram of the *Gastronomy Dataset*.

Event Dataset

The main classe in this dataset is **Event**, described by a *startDate*, an *endDate*, and a *description*. Every **Event** has an *organizer*, either a **Person** or an **Organization** and a *location*.

A **Person**—identified by *givenName*, *familyName*, *email*, and *telephone*—*worksFor* an **Organization**, which has a *name* and an *address*, i.e., a **PostalAddress** consisting of *streetAddress*, *postalCode*, *AddressLocality*, and *AddressCountry*.

Finally, an **Event** has as *location* a **MeetingRoom**—identified by a *name*— which is *containedInPlace* a **Place**—which has also a *name*

Figure 2.6: The UML diagram of the *Event Dataset*.

See also:

The *SPARQL howto*, which guides you in interacting with the SPARQL endpoint.

W3C Recommendation for [OWL2](#) and [RDF](#).

Official Specification of [UML Infrastructure](#) are available from [Object management group](#)

2.3.4 The AlpineBits Client



The AlpineBits Alliance strives to develop and to spread a standard format to exchange tourism data. There are two datasets they developed and keep up to date, that are of particular interest for Open Data Hub users: `HotelData` and `DestinationData`.

DestinationData

The [AlpineBits DestinationData](#) is a standardisation effort to allow the exchange of information related to mountains, events, tourism. Developed by the AlpineBits Alliance, Destination Data relies on a number of standards (Including *json*, REST API, Schema.org, OntoUML) to build the **AlpineBits DestinationData Ontology**, the core result of the effort. The goal of *DestinationData* it to provide a means to describe events, their location, and additional information on them. For this purpose, the DestinationData Ontology specifies a number of **Named Entities** used to describe *Events* and *Event Series*, *Mountain Areas*, *Places*, *Trails*, *Agents*, and so on.

The full specification of the ontology, including architecture of the API and description of the datatypes defines can be found in the latest **2021-04 version** of the official **Destination Data specs** ([pdf](#)).

The **reference implementation** of AlpineBits DestinationData is provided by Open Data Hub and publicly available at the dedicated endpoint at <https://destinationdata.alpinebits.opendatahub.bz.it/>.

See also:

More information and resources about AlpineBits DestinationData can be found on the [official page](#).

HotelData

The [AlpineBits HotelData](#) is meant for data strictly related to hotels and booking, like `Inventory Basic`, `Inventory HotelInfo`, and `FreeRooms`. This dataset can be access from the dedicated endpoint at <https://alpinebits.opendatahub.bz.it/AlpineBits/>

See also:

The dedicated howto *How to access Open Data Hub AlpineBits Server as a client*.

The [official page](#) of AlpineBits HotelData.

DOMAINS AND DATASETS

Changed in version 2022.03: Move Domain description in this section

What is a Domain?

A *Domain* is a category that contains entities that are closely related. In the Open Data Hub, each domain roughly identifies one social or economical category; the domains intended as sources for data served by the Open Data Hub are depicted at the bottom of [Figure 2.1](#).

Currently, the domains that can be accessed through the Open Data Hub are:

1. *Mobility*, this domain contains data about public transportation, parkings, charging station, and so on.
2. *Tourism*: data about events, accomodations, points of interest, and so on.
3. *Other* domain: a special category that encompasses domains that do not fall in any of the above category.

Each domain is composed by datasets, each of which contains data that provide useful information for the domain.



It is important to note that there is no clear separation between two domains. For example, data about public transportation belong to the Mobility domain, but are also useful for the Tourism domain.

The goal of the Open Data Hub project is to make available datasets containing data about the South Tyrolean ecosystem, to allow third parties to develop novel applications on top of them, consuming the exposed data. These applications may range from a simple processing of datasets to extract statistical data and to display the result in different graphic formats like pie-charts, to far more complex applications that combine data from different datasets and correlate them in some useful way.

As seen in [Figure 2.1](#), data originate from different domains (Mobility, Tourism, and so on); they are gathered from sensors and packed together by *Data Providers*. *Sensors* can be for example GPS devices installed on buses that send their real-time geographic position or a small electronic device on a plug of an e-charging station that checks the if the plug is being used or not, to let people know that the charging outlet is available.

Datasets are accessible through a *REST API*, the URL of each endpoint is given along with other information in the description of each dataset, see the lists of datasets in the remainder of this section.

3.1 Data Providers

A **Data Provider** is any entity that shares their Open Data with the Open Data Hub project, allowing their free reuse (ideally under a free licence like  or ) from any third-party that relies on the Open Data Hub to build their application. These entities can be private companies or enterprises, public bodies, and even private citizen, if they have interesting data about South Tyrol to share.


The Open Data exposed by the Open Data Hub originate from data and datasets owned by different actors (called **Data Providers**) which are at this time mostly local public bodies. Since there is no direct 1-to-1 correspondence between Data Providers and datasets, we currently offer a list of data providers whose data can be pulled from Open Data Hub. Indeed, an Open Data Hub dataset can be composed of data deriving from different providers, while a provider can submit to Open Data Hub multiple types of data that will belong to more than one dataset.



The Open Data Hub's Data Providers are:

- **Autostrada del Brennero/Brennerautobahn** management of the A22 motorway infrastructure
- **Alperia/Neogy** energy provider for South Tyrol
- **APPA Bolzano** South Tyrolean agency for the environment
- **APPA Trento** Trentino Agency of the environment
- **Bezirksgemeinschaft Burggrafenamt Comunità Comprensoriale Burgraviato**
- **Carsharing Alto Adige** via its technological partner DB Rent
- **CISMA** bluetooth sensors
- **IDM Südtirol/Alto Adige** trailblazer for economic development in South Tyrol
- **H2 Südtirol Alto Adige** energy company
- **HGV Hoteliers- und Gastwirteverband**
- **Inno.vie** mobility solutions
- **LTS South Tyrol** Association of Tourism Organisations
- **Municipality of Bolzano**
- **Municipality of Merano**
- **Municipality of Rovereto**
- **Municipality of Trento**
- **NOI Techpark** technology and science park of South Tyrol
- **Route220, Nevicam** and **Drive** e-charging stations provider
- **SASA** public transport operator
- **SIAG Südtirol Informatica AG** - Informatica Alto Adige
- **Südtirol Wein - Vini Alto Adige** consortium of South Tyrol Wines

3.2 Datasets, Open Data, and Licenses

The Open Data Hub contains many datasets: a few have been provided for testing purposes, other are meant for internal use only, and other contain only a part of their data that is available as Open Data.

While the goal of the Open Data Hub project is to expose **only Open Data** and the Open Data Hub team members always suggest to use  to third-parties releasing datasets, it is not yet possible for the Open Data Hub team to guarantee the availability as open data of all the data in the datasets, because the data licensing and its distribution rights are decided by the copyright holder of each dataset.

Since some of the datasets may contain data that can not be distributed by the Open Data Hub team under an open licence like, e.g.,  or , a user will be able to retrieve from each dataset only those data that are distributed as **Open Data**. The response to a query is in JSON format (although *CSV* output can be forced) and is **always** licensed as Open Data. However, the response may include resources like links to web pages, streams, or images that are subject to a different, even proprietary, licence. For more information about this topic, there is a *dedicated section in the appendices*.

At the date of writing, datasets mostly fall in either the *Mobility* and the *Tourism* domains; while a few more uncategorised datasets are available.

3.2.1 Authentication

The authentication layer is currently intended for **internal use only**. All data in the dataset that you can receive from the Open Data Hub are free to use and do not require any type of authentication.

The authentication layer can be of interest for developers who want to collaborate in the development of Open Data Hub; Details on the implementation are available in section *Authentication in the Open Data Hub*.

3.3 Datasets in the Mobility Domain

3.3.1 Technical Information for Mobility Dataset

This section contains detailed technical information shared by the datasets in the Mobility domain. The purpose of this section is manifold:

- To understand what output is received as result of a query and its structure—see *The JSON Response Schema*
- To know which methods are available to gather data from the domains via the API calls—see *Structure of the API calls and Payload*
- To learn how to improve responses by tweaking queries using SELECT, and WHERE clauses, logical operators, and parameters.—see *Advanced Data Processing*

The JSON Response Schema

We recall that every query to the mobility datasets will return a JSON-structured file with a number of information about one station (or more) and values it collected over time, both real-time and historical data.

The overall structure of the JSON is the following:

```
"offset": 0,  
"data": [],  
"limit": 200
```

Here, *offset* and *limit* are used for limiting the displayed results. The three keys have the following meaning:

- *limit* gives the maximum number of results that are included in the response. It defaults to **200**.

Hint: By setting the value to **-1**, *limit* will be disabled and all results will be shown.

- *offset* allows to skip elements from the result set. The default is **0**, i.e., the results start from the first one.
- *data* is the actual **payload** of the response, that is, the data answering the query; since it changes depending on which API call/method is used, it will be described in the next section.

Hint: It is possible to simulate pagination when there are many results: for example, if there are **1000** values, by adding to successive queries the offsets **0**, **200**, **400**, **600**, and **800**, the response of the query is split on 5 pages of 200 results each.

Structure of the API calls and Payload

In the Mobility domain, there are different general methods that can be used to extract data from the Open Data Hub's datasets and allow to incrementally refine the data retrieved. They are:

1. `/v2/` gives the list of the Open Data Hub's entry points, that is, the possible representations of the data contained in the datasets. to be used in the next methods. See [the details below](#).
2. `/v2/{representation}/` shows all the StationTypes available, that is, all the sources that provided data to the Open Data Hub.
3. `/v2/{representation}/{stationTypes}` returns data about the stations themselves, including metadata associated with each, and data about its parent stations, if any.
4. `/v2/{representation}/{edgeTypes}` returns data related to the [edges](#) and their parts, and is very similar to the previous call.
5. `/v2/{representation}/{stationTypes}/{dataTypes}`. In addition to the data of the previous call, it contains the data types defined in the dataset.
6. `/v2/{representation}/{stationTypes}/{dataTypes}/latest`. In addition to all the data retrieved by the previous call, this call retrieves also the most recent measurement. This method is especially suited for real time retrieval of data.
7. `/v2/{representation}/{stationTypes}/{dataTypes}/{from}/{to}`. All the data retrieved by method #3, but limited to a given historical interval (`from ... to`).

Note: The interval is *half-open*, i.e., [*from*, *to*), meaning that the *from* date is **included** in the result set, while the *to* date is **excluded**.

Representation types

The first method described in the previous list introduces the available entry points to the API v2: the types of *representation* that can be used to browse or access the data provided by the Open Data Hub Team

The *representation* consists now of a pair of comma-separated keywords composed of:

1. the already existent *flat* or *tree* AND
2. either *node* and *edge*

In both the **flat** and **tree** representations, all the metadata and available data are shown and browsable, the difference being that in *flat*, while *tree* keeps the hierarchical structure of the metadata.

The *node* and *edge* describe a **StationType** and the connection between two **StationTypes**, respectively.

Flat

In the *flat* representation, all metadata and available data can be accessed and browsed. However, no hierarchy appears and data and metadata are shown at the same level.

Tree

In the *tree* representation, all metadata and available data can be accessed and browsed as in *flat*, but in this case, any hierarchy of data or metadata is preserved and shown.

Node

A node is a measurement station and contains all metadata associated to it. The **node** representation corresponds to the *old* (pre-2020.10) output of the API calls, therefore it can safely be omitted for backward compatibility. As an example, valid for all methods listed in the [previous section](#), these API calls are equivalent.

```
/v2/tree,node/{stationTypes}
/v2/flat,node/{stationTypes}
/v2/tree/{stationTypes}
/v2/flat/{stationTypes}
```

Note: While only **available** nodes are exposed by the Open Data Hub, the resulting JSON response might still include the *savailable* field, short for station available.

Edge

An Edge is a connection between two stations, improved with additional information, including some descriptive field and geometries that describe the connection on a map. Internally, an edge is composed of three parts (all called *stations*): a start station (beginning of the edge), an end station and a station describing the edge. Whenever retrieving an Edge, all metadata referring directly to it begin with *e*, like for example *eactive*, *eavailable*, and so on.

Note: While only **available** edges are exposed by the Open Data Hub, the resulting JSON response might still include the *sbavailable*, *seavailable* and *eavailable* fields, referring to start station, end station, and edge description, respectively.

Moreover, there are neither measurements nor types associated with edges.

Valid combinations are therefore: *flat,node*; *tree,node*; *flat,edge*; *tree,edge*; if neither *node* or *edge* are provided, the default **node** will be used.

An additional representation is *apispec*, which allows to see and reuse the API specification in an OpenAPI v3 YAML format, suitable for swagger-like access to the data.

In the reminder of this section we show examples of some of the above mentioned API methods and describe the outcome, including the various keys and types of data returns by the call.

`/v2/{representation}/{stationTypes}`

To describe the outcome of this method in details, we will use the following snippet.

Listing 3.1: An excerpt of information about a charging station.

```
1  {
2  "pactive": false,
3  "pavailable": true,
4  "pcode": "AER_000000005",
5  "pcoordinate": {
6    "x": 11.349217,
7    "y": 46.499702,
8    "srid": 4326
9  },
10 "pmetadata": {
11   "city": "BOLZANO - BOZEN",
12   "state": "ACTIVE",
13   "address": "Via Cassa di Risparmio - Sparkassenstraße 14",
14   "capacity": 2,
15   "provider": "Alperia Smart Mobility",
16   "accessType": "PUBLIC",
17   "paymentInfo": "https://www.alperiaenergy.eu/smart-mobility/punti-di-ricarica.html",
18   "municipality": "Bolzano - Bozen"
19 },
20 "pname": "BZ_CASSARISP_01",
21 "porigin": "ALPERIA",
22 "ptype": "EChargingStation",
23 "sactive": false,
24 "savailable": true,
25 "scode": "AER_000000005-1",
26 "scoordinate": {
27   "x": 11.349217,
28   "y": 46.499702,
29   "srid": 4326
30 },
31 "smetadata": {
32   "outlets": [
33     {
34       "id": "1",
35       "maxPower": 22,
36       "maxCurrent": 31,
37       "minCurrent": 0,
```

(continues on next page)

(continued from previous page)

```

38     "hasFixedCable": false,
39     "outletTypeCode": "Type2Mennekes"
40   }
41 ],
42   "maxPower": 7015,
43   "maxCurrent": 31,
44   "minCurrent": 6,
45   "municipality": "Bolzano - Bozen",
46   "outletTypeCode": "IEC 62196-2 type 2 outlets (all amperage and phase)"
47 },
48 "sname": "BZ_CASSARISP_01-253",
49 "sorigin": "ALPERIA",
50 "stype": "EChargingPlug"
51 }

```

You immediately notice that all the keys in the first level start either with a **p** (*p*active, *p*coordinate, and so on) or an **s** (*s*active, *s*coordinate, and so on): the former, **p**, refers to data about the *parent* stations, **s** to data of the station itself. Besides the initial *p* or *s*, the meaning of the key is the same. In the snippet above, you see that all the data about a station are grouped together and come after the data of its parent (see lines.

The meaning of the keys are:

- **active**: the station is actively sending data to the Open Data Hub. A station is automatically marked as not active (i.e., `pactive = false`) when it does not send data for a given amount of time (24 hours).
- **available**: data from this station is available in the Open Data Hub.

Note: *active* and *available* might seem duplicates, but a station can be available but not active or vice-versa: In the former case, it means that its historical data have been recorded and can be accessed, although it currently does not send any data (for example, due to a network error or because it is not working or because it has been decommissioned); in the latter case, the station has started to send its data but they are not yet accessible (for example, because they are still being pre-processed by the Open Data Hub).

- **code**: a unique **ID**entifier
- **coordinate**: the station's geographical coordinates
- **metadata**: it may contain any kind of information about the station and mostly depends on the type of the station and the data it sends. In the snippets above, lines 10-16 contain information about the location of a charging station, while lines 28-38 technically describe the type of plugs available to recharge a car.

Hint: The metadata has only one limitation: it must be either a JSON object or NULL.

- **name**: a (human readable) name of the station
- **origin**: the *source* of the station, which can be anything, like for example the name of the *Data Providers*, the spreadsheet or database that contained the data, a street address, and so on.
- **type**: the type of the station, which can be a *MeteoStation*, *TrafficStation*, *EChargingPlug*, *Bicycle*, and so on.

Note: The name of the StationType is **Case Sensitive**! You can retrieve all the station types with the following API call.

```
~$ curl -X GET "https://mobility.api.opendatahub.bz.it/v2/tree" -H "accept: application/json"
```

/v2/{representation}/{stationTypes}/{dataTypes}/latest

This API call introduces two new prefixes to the keys, as shown in [Listing 3.2](#).

Listing 3.2: An excerpt of information about a charging station.

```

1  {
2      "tdescription": "",
3      "tmetadata": {},
4      "tname": "number-available",
5      "ttype": "Instantaneous",
6      "tunit": "number of available vehicles / charging points",
7
8      "mperiod": 300,
9      "mtransactiontime": "2018-10-24 01:05:00.614+0000",
10     "mvalidtime": "2020-05-01 07:30:00.335+0000",
11     "mvalue": 1,
12 }

```

The new prefixes are **t** and **m**. The *t* prefix refers to **Data Types**, i.e., how the values collected by the sensors are measured. See below for a more detailed description of data types and some tip about them. The *m* prefix refers to a **measurement**, that is, how often the data are collected, timestamp of the measure, when it is transmitted to be stored, and other information.

Alongside all keys present in Listing 3.1 (see *previous section*), Listing 3.2 contains the additional key:

- **ttype**: the type of the data, which can be expressed as either a custom string, like in the example above, or as a DB function like COUNT, SUM, AVERAGE, or similar
- **tunit** the unit of measure
- **mperiod**: the time in seconds between two consecutive measures
- **mtransactiontime**: timestamp of the transmission of the data to the database
- **mvalidtime**: timestamp of the measurement. It is either the moment in time when the measurement took place or the time in the future in which the next measure will be collected.
- **mvalue**: the absolute value of the measure, represented in either *double precision* or *string* format. It must be paired with the t keys to understand its meaning.

Listing 3.2 represents an *EChargingStation* with one available charging point; the last measure was taken on 2020-05-01 07:30:00.335+0000 and will be repeated every 5 minutes (300 seconds). Moreover, the station appears to not transmit its data anymore, so historical data might not be available.

Data types in the datasets.

Data types are not normalised; that is, there is no standard or common unit across the datasets. Indeed, each data collector defines its own data types and they may vary quite a lot from one dataset to another. There is also neither a common representation format for data types, therefore a same unit can appear quite different in different datasets. For example, to express *microseconds*, one dataset can use

```
"tdescription": "Time interval measured in microseconds",
"tmetadata": {},
"tname": "Time interval",
"ttype": "Instantaneous",
"tunit": "ms",
```

While another:

```
"tdescription": "Microseconds between two consecutive measures",
"tmetadata": {},
"tname": "Time interval",
"ttype": "COUNT",
"tunit": "milliseconds",
```

We can see that, although we might understand that the measures from the two datasets are indeed expressed in milliseconds, this is not true for machine-processed data

`/v2/{representation}/{stationTypes}/{dataTypes}/{from}/{to}`

This method does not add any other keys to the JSON response; all the keys described in the previous two section are valid and can be used.

Advanced Data Processing

Before introducing advanced data processing techniques, we recall that queries against the Open Data Hub's datasets always return a **JSON** output.

Advanced processing allows to build SQL-style queries using the **SELECT** and **WHERE** keywords to operate on the JSON fields returned by the calls described in the previous section. **SELECT** and **WHERE** have the usual meaning, with the former retrieving data from a JSON field, in the form of **SELECT=target[,target,...]**, and the latter retrieving records from the JSON output, using the **WHERE=filter[,filter,...]** form, with an implicit **and** among the filters, therefore evaluation of the filters takes place only if all filters would individually evaluate to **true**.

The SELECT Clause

In order to build select clauses, it is necessary to know the structure of the JSON output to a query, therefore we illustrate this with an example with the following excerpt from the *parking dataset* that represents all data about one parking station:

```
{
  "sactive": false,
  "savailable": true,
  "scode": "102",
  "scoordinate": {
    "x": 11.356305,
    "y": 46.496449,
    "srid": 4326
  },
  "smetadata": {
    "state": 1,
    "capacity": 233,
```

(continues on next page)

(continued from previous page)

```
"mainaddress": "Via Dr. Julius Perathoner",
"phonenumber": "0471 970289",
"municipality": "Bolzano - Bozen",
"disabledtoiletavailable": true
},
"sname": "P02 - City parking",
"sorigin": "FAMAS",
"stype": "ParkingStation"
}
```

You see that there are two hierarchies with two levels in the snippet: *scoordinate* and *smetadata*; to retrieve only data from them we will use the *select* clause with the `/v2/{representation}/{stationTypes}` call; you can therefore:

- retrieve only the metadata associated with all the stations; the select clause would be: `select=smetadata`
- retrieve all the cities in which there are ParkingStations with `select=smetadata.municipality`
- retrieve all cities and addresses of all ParkingStations: `select=smetadata.municipality,smetadata.mainaddress`

The latter two examples show that to go down one more step into the hierarchy, you simply add a dot (".") before the attribute in the next level of the hierarchy. Moreover, you can extract multiple values from a JSON output, provided you separate them with a comma (",") and use **no empty spaces** in the clause. In the above examples, each of the element within parentheses--`smetadata`, `smetadata.municipality`, and `smetadata.mainaddress`-- is called **target**.

Within a **SELECT** clause, SQL functions are allowed and can be mixed with targets, allowing to further process the output, with the following limitations:

- Only *numeric* functions are allowed, like e.g., `min`, `max`, `avg`, and `count`
- **No** string selection or manipulation is allowed, but left as a post-processing task
- When a function is used together with other targets, these are used for grouping purposes. For example: `select=sname,max(smetadata.capacity),min(smetadata.capacity)` will return the parking lots with the highest and lowest number of available parking spaces.

The WHERE Clause

The **WHERE** clause can be used to define conditions to filter out unwanted results and can be built with the use of the following operators:

- *eq*: equal
- *neq*: not equal
- *lt*: less than
- *gt*: greater than
- *lteq*: less than or equal
- *gteq*: greater than or equal
- *re*: regular expression
- *ire*: case insensitive regular expression
- *nre*: negated regular expression
- *nire*: negated case insensitive regular expression

- *bbi*: bounding box intersecting objects (ex., a street that is only partially covered by the box)
- *bbc*: bounding box containing objects (ex., a station or street, that is completely covered by the box)
- *in*: true if the value of the target can be found within the given list. Example: *name.in.(Patrick,Rudi,Peter)*
- *nin*: False if the value of the target can be found within the given list. Example: *name.nin.(Patrick,Rudi,Peter)*
- *and(filter,filter,...)*: Conjunction of filters (can be nested)
- *or(filter,filter,...)*: Disjunction of filters (can be nested)

As an argument to the *filter*, it is possible to add either a single value or a list of values; in both cases, operators are used to determine a condition and only items matching all of the filters will be included in the answer to the query (implicit *AND*). Like in the case of *SELECT* clauses, multiple comma-separated conditions may be provided. As an example, the following queries use a value and a list of values, respectively:

- `where=smetadata.capacity.gt.100` returns only parking lots with more than 100 parking spaces
- `where=smetadata.capacity.gt.100,smetadata.municipality.eq."Bolzano - Bozen"` same as previous query, but only parking lots in Bolzano are shown.

In these two examples we use a number in the filter (i.e., `gt.100`), which is by default automatically recognised as a number and the required math is calculated out of the box. In case there is a query in which you use a number, but need to consider it as a string, you need to use double quotes, like `gt."100"`.

Logical Operators

Besides the operators described in section *The WHERE Clause*, Open Data Hub supports the use of logical operators *and* and *or* in the *WHERE* clause, like these examples show.

```

1 and(x.eq.3,y.eq.5)
2 x.eq.3,y.eq.5
3
4 or(x.eq.3,y.eq.5)
5 or(x.eq.3,and(y.gt.5,y.lt.10))

```

Logical operators are followed by a comma-separated list of *targets*, which can be filters (see previous section for some example), or other logical operators. In complex logical expression, parentheses are employed to assign precedence. Lines 1 and 2 above are equivalent, because the default logical operator is *and*.

The above example will be translated into Postgres as follows:

```

1 (x = 3 AND y = 5)
2 (x = 3 AND y = 5)
3
4 (x = 3 OR y = 5)
5 (x = 3 OR (y > 5 AND y < 10))

```

Additional Parameters

There are a couple of other parameter that can be given to the API calls and are described in this section.

shownull

In order to show **null** values in the output of a query, add `shownull=true` to the end of your query.

distinct

Results in query responses contain unique results, that is, if for some reason one element is retrieved multiple times while the query is executed, it will be nonetheless shown only once, for performance reasons. It is however possible to retrieve each single result and have it appear in the response by adding `distinct=true` to the API call.

Warning: Keeping track of all distinct values might be a resource-intensive process that significantly rises the response time, therefore use it with care.

timezone

By default, the timestamp of the Open Data Hub responses is given in **UTC** time zone. The use of the `timezone` parameter allows to modify the timestamp whenever desirable. To use it, simply append the parameter to your API call.

```
/flat/ParkingStation/occupied/latest?timezone=UTC-2
```

```
/flat/ParkingStation/occupied/latest?timezone=Europe/Rome
```

Note: As argument to the `timezone` parameter, you can use any allowed value in [Java's Time zone implementation](#).

This section contains *technical information* about the datasets in the Mobility Domain and how to access them using the API that the Open Data Hub team developed and made available.

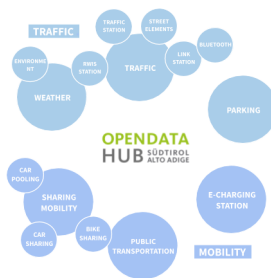


Figure 3.1: The dataset in the Mobility Domain at a glance.

Note: Recall that the API v1 for the Mobility Domain is now **deprecated**.

The description of each dataset includes the following information:

Output	The output format of the API call
E-mail contact	An e-mail contact for the dataset
API version	The versions of the API that can be used to access dataset
StationType	The direct link to each stationType included in the dataset
Use cases and info	Link to web sites that use the dataset and to use cases based on the dataset
Web component	Link to Web Components developed on top of the dataset (optional)
Sources	The list of Data Providers whose data compose the dataset

Note: There is one StationType, namely **MobileStation** which is a mobile probe no longer active. It will always return an empty set of values, because historical data are not available in the Open Data Hub.

The datasets in the Mobility domain are grouped in **Traffic** and **Mobility** sub-domains as follows:

See also:

The following howto will help you access data in the Mobility domain:

How to Access Mobility Data With API v2 Access and technical details about the available data

Other howtos are available in the *dedicated section*.

3.3.2 Traffic

The Mobility/Traffic sub-domain contains data about traffic (like e.g., real time traffic load of a street, environmental measurement) that are useful to plan a trip with an own means of transport, for example a car, or a bike.

it.bz.opendatahub.bluetooth

The data for this datasets are collected by experimental Bluetooth-based sensors and detectors currently located on various points of the streets of Bolzano and soon in other location of South Tyrol. Gathered data are then processed to obtain useful information about the traffic; therefore, data in this dataset are:

- The total number of vehicles detected
- An estimation of heavy and light vehicles

Collected data are also split within intervals (of e.g., 15, 30 minutes), for statistical and historical offline analysis. Moreover, the data gathered by the Bluetooth devices are used in the *linkstation dataset*.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	BluetoothStation
Use cases and info	https://analytics.opendatahub.bz.it/
Sources	CISMA

it.bz.opendatahub.environment

In this dataset can be found **Pollution** and **Air Quality data** (including, but not limited to, CO, NO2, PM 2.5, PM10). Some portion of the data have been manually elaborated, with the purpose to increase data quality.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	EnvironmentStation
Use cases and info	https://analytics.opendatahub.bz.it/ , Project STODT
Sources	APPA Bolzano, APPA Trento, A22 Municipality of Bolzano

it.bz.opendatahub.linkstation

The data for this dataset are collected by the same Bluetooth-based sensors that are used to produce the *Bluetooth dataset*. Indeed, the data gathered by the sensors will be used to produce statistics about a LinkStation, which is defined as the path between an ordered pair of bluetooth stations.

- the **valid matches** between pairs of Bluetooth sensors (also called Bluetooth stations).
- The **number of matches per predefined time interval** (e.g., 30 or 60 minutes), like in the Bluetooth dataset
- The **estimated travel time and speed** of the vehicle, computed for every 15 minutes interval only.

The definitions and algorithm used in the computations are extensively described in Section *Data analysis complexity of this pdf article*.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	LinkStation
Use cases and info	https://analytics.opendatahub.bz.it/
Sources	A22

it.bz.opendatahub.parking

This dataset contains two types of data:

- Parking stations - off street parking data (currently for the cities of Bolzano, Merano, Trento, Rovereto), with slot availability and predictions
- Parking sensors - on street parking data (currently Bolzano, soon Merano). Single parking slots on streets, can be within a virtual area

Output	JSON, mime-type application/json		
E-mail contact	contact		
API version	v1 deprecated, v2		
StationType	ParkingStation, ParkingSensor		
Use cases and info	https://parking.bz.it	https://mobility.meran.eu	https://analytics.opendatahub.bz.it
	https://mobility.bz.it		
Sources	Municipalities of Bolzano, Merano, Trento and Rovereto		

it.bz.opendatahub.rwisstation

The **Road Weather stations** are sensors that measure both the *road surface* and the *weather conditions*. Data represent a showcase from project CLEAN-ROADS, which offered open data until 2016.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	RWISstation
Use cases and info	https://analytics.opendatahub.bz.it/ , https://map.clean-roads.eu/
Sources	A22

it.bz.opendatahub.streetelements

This datasets contains historical data about air quality processing along the roads.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	Streetstation
Use cases and info	–
Sources	CISMA

it.bz.opendatahub.trafficstation (1)

This datasets contains aggregated data collected by sensors drowned in asphalt, but separated for each lane of a street. For people that travel on Italian streets, these sensors are those large rectangle boxes engraved in asphalt that you can see also on streets in the cities.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	traffic, TrafficSensor, Trafficstation, TrafficStreetFactor
Use cases and info	https://analytics.opendatahub.bz.it/
Sources	Municipality of Bolzano, A22

it.bz.opendatahub.trafficstation (2)

The **VMS** (Variable Message Sign) sensors collect data about the electronic road signs on the A22 Motorway.

This dataset is formally part of the *traffic station dataset*, but since data are specific to VMS, we split them for clarity.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	VMS
Use cases and info	https://analytics.opendatahub.bz.it/
Sources	A22

it.bz.opendatahub.weather

This dataset contains meteorological data provided by the *Weather and Avalanche Services* of South Tyrol and Trentino, such as Precipitation and Air temperature

This dataset contains the **measurements** of the weather data, not the **forecast**.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	MeteoStation
Use cases and info	https://analytics.opendatahub.bz.it/
Sources	Weather and Avalanche Services of South Tyrol and Trentino

3.3.3 Mobility

The Mobility/Mobility sub-domain contains data about public transportation, sharing of transport means, and recharging stations for e-cars.

[it.bz.opendatahub.bikesharing](#)

This dataset contains data of the **bike sharing** service in Bolzano and Merano, such as stations (for station based service - Bolzano), areas (for free floating service - Merano), availability, battery state, and other useful information.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	Bicycle BikesharingStation Bicyclestationbay
Use cases and info	https://mobility.merano.eu/ https://analytics.opendatahub.bz.it/
Sources	Municipalities of Bolzano and Merano

[it.bz.opendatahub.carpoolinghub](#)

This dataset contains all data of the carpooling service of the Burggrafenamt/Burgraviato region in South Tyrol, including carpooling users, car availability, and locations with accessible destinations.

Note: Only users that have explicitly expressed their consent to privacy are included, no sensitive data is stored or exchanged.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	CarpoolingHub CarpoolingService CarpoolingUser
Use cases and info	https://carpooling.bz.it/ https://mobility.merano.eu/ https://analytics.opendatahub.bz.it/
Sources	Bezirksgemeinschaft Burggrafenamt/Comunità Comprensoriale Burgraviato, Inno.vie

[it.bz.opendatahub.carsharing](#)

This dataset contains all data of the **car sharing** service in South Tyrol, including stations and availability.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	CarsharingStation
Use cases and info	https://mobility.merano.eu/ https://mobility.bz.it/ https://analytics.opendatahub.bz.it/
Sources	Car Sharing Südtirol Alto Adige

it.bz.opendatahub.echargingstation

This dataset exposes data about the existing e-charging stations and plugs in South Tyrol and their status, including historical data and usage. Data about station offering **Hydrogen** recharged are also included.

Note: For this dataset are available also a number of *Web Components*, see the use cases.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1 deprecated, v2
StationType	EChargingPlug EChargingStation
Use cases and info	https://mobility.meran.eu/ https://analytics.opendatahub.bz.it/ https://mobility.bz.it/
Web Components	Smart Mobility Mobility E-Charging Map Mobility E-Charging Dashboard
Sources	Alperia (Neogy), Route220, nevicam, Driwe, H2 Südtirol Alto Adige

Public Transportation

Warning: This dataset is not supported anymore and will be removed from the Open Data Hub in the next future.

This dataset shows the real time position of buses operated by SASA in South Tyrol and, through a few subsets, additional information about lines, station boards, and news.

There are additional subsets that expose data in different formats:

- info.opensasa.plandata (VDV 451 - VDV 452)
- info.opensasa.stationboard (JSON)
- info.opensasa.news (JSON)
- info.opensasa.rssDE (XML)
- info.opensasa.rssIT (XML)

Output	geoJSON
E-mail contact	info@sasabus.org
API version	–
URL	https://daten.buergernetz.bz.it/dataset/southtyrolean-public-transport https://sasabus.org/opendata
Use cases and info	https://suedtirolmobil.info/en/ , https://mobility.meran.eu/ , https://analytics.opendatahub.bz.it/ , https://mobility.bz.it/
Web Component	Smart Mobility
Sources	STA, SASA

3.4 Datasets in the Tourism Domain

New in version 2012.12: New WeatherHistory dataset

3.4.1 Technical Information for Tourism Dataset

This section contains detailed technical information shared by the datasets in the Tourism domain. The purpose of this section is manifold:

- To know which methods are available to gather data from the domains via the API calls—see *Structure of the API calls*
- To learn how to improve responses by adding filters to the queries.—see *Filters common to all datasets*
- Which kind of data are accepted by the API methods—see *Types of input data*

Structure of the API calls

In the Tourism domain, there are a few API calls that allow to extract the same type of data from the various datasets. Each of these calls can prove useful in different scenarios, depending on the data returned and is described in this section, in which the following conventions are used:

- {Name} is the (case sensitive!) name of the dataset you are currently working with, like for example `Accommodation`.
- {Id} is the unique identifier of an array within the dataset, i.e., an item of the dataset. It is usually the first key of the resulting JSON output of a query.

The calls defined for every datasets are:

- `/api/{Name}` Return the whole dataset.
- `/api/{Name}/{Id}` Return only item with given {Id}.
- `/api/{Name}Reduced` Return only the list of Ids and respective name of the items in the dataset. It is useful to create lists of items or just to have an overview of the dataset's items.
- `/api/{Name}Changed` Return all items that have changed since date YYYY-MM-DD
- `/api/{Name}Types` Returns all types of data present in the dataset, that can be later used to ask more precise queries to the dataset.

The following calls have been **removed** and can not be used anymore. They have been replaced by a new filter, called *language*, that operates on the datasets in a similar way to the *The fields Filter* and is described in section *The language Filter*.

- ~~`/api/{Name}Localized` Return the whole dataset in only the given language (which is a mandatory part of the query)~~
- ~~`/api/{Name}Localized/{Id}` Return only item with given Id in given language.~~

Filters common to all datasets

Note: Besides the filters available globally, for each dataset several additional filters are available. They are described in the respective swagger interface.

Filters are used within a dataset and their primary purpose is to limit the result set according to specific parameters, although they might not be available in every API call. Information about default values can be found for each datasets in the [swagger interface](#) of the API. Some examples of their use can be found in section [Quick and \(not-so\) Dirty Tips for Tourism \(AKA Mini-howtos\)](#).

- **Seed** is used to set pagination. See tip [TT3](#).
- **Locfilter** is a composed parameters that uniquely identifies a location within South Tyrol. See example [EX2](#) for a detailed example.
- **Latitude** and **Longitude** are used to identify the (absolute) positioning of a location, point of interest, event, or any other type of object. They must be entered in decimal form
- **Radius** it is the distance in meter from a geographical point. It can be used together with latitude and longitude to broaden the search for an object. The results are automatically *geosorted*, that is, they are listed from the nearest to the most far away from the selected point. The distance is calculated as the crow flies.
- **IdFilter** allows to extract from the dataset only the items with the given IDs, separated with a , .
- **Active** and **OdhActive**. Filters with the same name, with one prefixed by **Odh** refer to the same parameter. The difference is however important: **Active** indicates that the item is present in the original dataset provided, while **OdhActive** shows that the item has been verified by the Open Data Hub team and is present in the Open Data Hub. See discussion in tip [TT2](#).
- **ODHTag** allows to filter a result set according to tag defined by the Open Data Hub team. These tags are mostly related with places to see, activities that can be carried out in winter or summer, food and beverage, cultural events and so on

Special common filters

This section describes some useful filters that can be used on all Tourism Datasets. Some of them relies on simpler filters, like *field*, that is described in the [The fields Filter](#) section below. These filters allow to customise queries and have been introduced for all cases for which there is no existent filter or sorting possibilities.

`rawfilter`

rawfilter can be appended to any query with the syntax `?rawfilter=<filter(s)>`, in which `<filter>` has the generic form `<field>`, `<value>`. These logical operators can be used to combine multiple filters: *eq*, *ne*, *gt*, *ge*, *lt*, *le*, *and*, *or*, *isnull*, *isnotnull*, *in*, *nin*

`rawsort`

rawsort can be used to sort in ascending order the results of a query; its syntax is `?rawfilter=<filter(s)>`. Here, `<filter>` is the name of a field in the result set. Multiple fields can be specified as comma separated, e.g., `?rawfilter=startDate,Detail.en.Title`. If a `<filter>` is prefixed with a dash, - sorting is reverted, i.e., output is shown in descending order.

`removenullvalues`

`?removenullvalues=true` removes all **NULL** values from the query's output. While usually it's always desirable to have a full JSON output to be parsed, removing NULL values proves useful to reduce the output size or to verify data quality. By using `removenullvalues`, one can check if all fields of a given entry are populated or not.

The *fields* Filter

A recently added filter is the **fields** filter, which allows to add to a REST request a parameter that can act on multiple keys of a dataset entry, selecting only the entries which have a corresponding value in the dataset. In other words, the purpose of this filter is to retrieve only relevant information from each item in the datasets and strip down information that is not needed or not necessary to the purpose of the query. The *fields* filter can be used on single-valued parameters as well as on dictionary fields.

Lets take as example the *ODHActivityPOI* dataset and its swagger interface <https://tourism.api.opendatahub.bz.it/#/ODHActivityPoi>; the same approach can be used with other datasets by simply replacing the datasets' name in the URL.

The following query will retrieve from the dataset only those item which have a **Type** and a strong:Active keys defined in the dataset:

```
https://tourism.opendatahub.bz.it/api/ODHActivityPoi?fields=Type,Active
```

The following query retrieves information from within a dictionary field:

```
https://tourism.opendatahub.bz.it/api/ODHActivityPoi?fields=Detail.en.Title
```

In particular, all items which have a *Title* in english within the *Detail* will appear in the result set of this query.

To show how it works, the following excerpt from the dataset shows how to discover the **Detail.en.Title** elements:

```
"Detail": {  
  "en": {  
    "Title": "01 Cross Country Stadio Track Dobbiaco/Toblach",  
    "Header": null,  

```

The *language* Filter

The *language* filter can be seen as a special case of the more generic *fields* filter, described in the previous section, and is similar to the second example presented there.

The *language* filter is used to retrieve only the data stored in one of the languages supported by the Open Data Hub. Let's build on the example of previous section and use the *ODHActivityPOI* dataset. The following query will retrieve all the data in the dataset that have some information stored in English:

```
https://tourism.api.opendatahub.bz.it/v1/ODHActivityPoi?language=en
```

Most of the data in the Open Data Hub datasets are available in three languages, English, German, and Italian, for which **en**, **de**, and **it** can be used as value of the *language* filter. Additional language in which data may be available are: Dutch (**nl**), Czech (**cs**), Polish (**pl**), French (**fr**), and Russian (**ru**).

The search Filter

Currently available for only a limited number of datasets, namely Accommodations, Gastronomies, Events, Activities, Pois, ODHActivitiesPois, and Article, this filter allows to find whether the given string is contained in one of the field of the JSON response sent as answer to a query.

Exporting and saving data

Queries to the Open Data Hub datasets always return data in JSON format and can be saved in that format either from the browser or from the CLI, in the latter case by simply piping the output to a file. Additionally, it is now possible to save data also in CSV (Comma Separated value) format.

Warning: This feature is currently available only for the following datasets:

Accommodation, Activity, Article, District, Event, Gastronomy, MetaRegion, Municipality, ODHActivityPoi, Poi, Region, SkiArea, SkiRegion, and TourismAssociation

However, plans are to soon have all Tourism datasets support it.

Depending on how you access the data, there are different modalities to retrieve and save data in CSV format:

- when using a browser, append the keyword `&format=csv` to any query and you will be prompted to provide a name to the file that will contain the required data. Example:

```
https://tourism.api.opendatahub.bz.it/v1/Activity?fields=Id,Detail.en.Title,ContactInfos.en.
CompanyName&pagesize=500
```

This query shows its JSON output on the screen. To save it, right click on the page and select *Save as*.

```
:apit:`Activity?fields=Id,Detail.de.Title,ContactInfos.de.CompanyName&pagesize=500&
↪format=csv`
```

Nothing is shown on screen, but a dialog window opens that allows you to select a name for the file and the directory where to save it.

- When using a CLI command to query the Tourism endpoint, replace the header that you send with the **curl** command:

```
~$ curl -X GET "https://tourism.api.opendatahub.bz.it/v1/Activity?fields=Id,Detail.
↪en.Title,ContactInfos.en.CompanyName&pagesize=500" -H "accept: application/json"
```

The output of this query will be in JSON format.

```
~$ curl -X GET "https://tourism.api.opendatahub.bz.it/v1/Activity?fields=Id,Detail.
↪en.Title,ContactInfos.en.CompanyName&pagesize=500" -H "accept: text/csv"
```

The output of this query will be in CSV format.

- When using an API Development Environment like Postman, add *accept: text/csv* to the Header of the request. See detailed procedure and screenshot can be found in the [Data Exporting](#) section of Postman's howto.

Types of input data

Since calls in the tourism domain are quite generic and revolve around a few common calls (see section *Structure of the API calls*), we showed a couple of filters that can be used to reduce the result set and make the query more precise. Depending on the type of filter, a different type of data must be entered to have a successful result, otherwise the filter will not match. In this section we show the most common types of data that should be provided, besides the common strings, dates, and integers.

Bitmask value

A Bitmask value is a kind of shorthand that can be entered in a filter to obtain results for different types of that filter's accepted values. Each of the accepted values has a code that is a power of two (1, 2, 4, 8, and so on), hence each sum of different codes produces a unique number. The advantage is that, instead of entering multiple strings that should be matched, you simply need to enter a number as a filter, that is the sum of the values' corresponding codes. See *Example 3*.

Lists

A list is an (unordered) sequence of items. The available values are usually listed on the right-hand side of the filter, along with the separator, which is a **comma** (.). In a few cases, in which more lists are accepted as filter.

Compound values

Compound values refer to those values that need a prefix before the type of value. See for example *Example2* for a deeper explanation and *Example 1* for a sample query that fails because a wrong compound value was supplied.

Language

The descriptions of items in the dataset appear in three languages: Italian, German, and English. To retrieve values only in one language, enter **it**, **de**, or **en**, respectively.

This section contains *technical information* about the dataset in the Tourism Domain and how to access them using the API that the Open Data Hub team developed and made available.

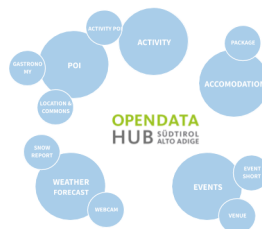


Figure 3.2: The dataset in the Tourism Domain at a glance.

Datasets presented here are related to all kind of touristic activities in South Tyrol. By exploring this domain, it is possible to find information about winter and summer offers from local touristic boards, information about weather, hotels and accommodation, Points of Interests, and a lot more.

The following information is provided for each dataset in the Tourism domain:

Output	The output format of the API call
E-mail contact	An e-mail contact for the dataset
API version	The versions of the API that can be used to access the dataset
Swagger URL	The URL of the swagger interface to the data
API URL	The URL of the browsable version of the dataset
Use cases and info	Link to web sites that use the dataset and to use cases based on the dataset
Android App	Link to app for mobile phones developed using the data in the dataset
Sources	The list of Data Providers whose data compose the dataset
SPARQL Endpoint	Dataset is accessible through the SPARQL Endpoint ¹

See also:

The following howto will help you access data in the Tourism domain:

How to access Tourism Data? Access and technical details about the available data

How to use the Open Data Hub's Tourism Data Browser? Browse Open Data offered by the Open Data Hub

Quick and (not-so) Dirty Tips for Tourism (AKA Mini-howtos) Quick tips and troubleshooting

Other howtos are available in the *dedicated section*.

it.bz.opendatahub.accommodation

This dataset contains various data about **accommodation** in South Tyrol, including information about the rooms.

Besided with Open Data Hub API, this dataset can be queried with the AlpineBits protocol.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1, AlpineBits HotelData 2017-10, 2018-10, and soon 2020-10
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Accommodation
API URL	https://tourism.api.opendatahub.bz.it/v1/Accommodation
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Android App	Südtirol Guide/Alto Adige Guide
Sources	LTS
SPARQL Endpoint	https://sparql.opendatahub.bz.it

it.bz.opendatahub.activity

This dataset contains data about activities from LTS, including local tours, hiking, running & fitness, cross-country skiing, alpine climbing, e-bike, downhill, nordic walking, and many more. This dataset offers LTS categorization and enhanced filtering.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Activity
API URL	https://tourism.api.opendatahub.bz.it/v1/Activity
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Sources	LTS ActivityData

it.bz.opendatahub.activity_poi

This dataset contains a collection of activities and Points of Interest (**PoI**) in the South Tyrol region. The available data have been extracted from different sources and also offer IDM categorisation. This is a kind of *superdataset*, which includes also *poi dataset*, *activity dataset*, and *gastronomy dataset*.

¹ This information is provided only if the dataset is accessible through SPARQL.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/ODHActivityPoi
API URL	https://tourism.api.opendatahub.bz.it/v1/ODHActivityPoi
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Android App	Südtirol Guide/Alto Adige Guide
Sources	LTS ActivityData, LTS PoiData, LTS GastronomicData, SuedtirolWein, SIAG Museum data, IDM Content, and other Sources

it.bz.opendatahub.common

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Common

it.bz.opendatahub.event

This dataset contains data about active events provided by LTS (TIC Web): meetings, fairs, markets, performances, courses, music, festivals, hikes, trips, guided tours, exhibitions, and so on. Data about past events is also included.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Event
API URL	https://tourism.api.opendatahub.bz.it/v1/Event
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Android App	Südtirol Guide/Alto Adige Guide
Sources	LTS
SPARQL Endpoint	https://sparql.opendatahub.bz.it

it.bz.opendatahub.eventshort

This dataset contains events provided by **NOI Techpark**, **Eurac** and **St. Virtual**.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/EventShort
API URL	https://tourism.api.opendatahub.bz.it/v1/EventShort
Use cases and info	https://today.noi.bz.it/ https://noi.bz.it/it/event-calendar/ https://virtual.noi.bz.it/programma.html
Sources	LTS

it.bz.opendatahub.gastronomy

This dataset contains data about gastronomy locations from LTS, including restaurants, bars, bistros, pubs, apres ski, pizzerias, fast food, cafeterias, vinotheques, beer gardens, mountain refuges, alpine huts, ski huts, etc. This dataset offers enhanced filtering (cuisine types, ceremony codes, dish codes etc.).

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Gastronomy
API URL	https://tourism.api.opendatahub.bz.it/v1/Gastyronomy
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Sources	LTS
SPARQL Endpoint	https://sparql.opendatahub.bz.it

it.bz.opendatahub.location

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Location
API URL	https://tourism.api.opendatahub.bz.it/v1/Location

it.bz.opendatahub.package

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://service.suedtirol.info/swagger/ui/index#/Package

it.bz.opendatahub.poi

This dataset contains data about PoIs from LTS, such as beauty and wellness centres, bikes, castles, climbing halls, cocktail bars, e-bike rentals, fashion shops, pharmacies, gas stations, car parks, shops, ski schools, and many more. This dataset offers LTS categorisation.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Poi
API URL	https://tourism.api.opendatahub.bz.it/v1/Poi
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Sources	LTS PoiData

it.bz.opendatahub.ski

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#!/Common/get_v1_SkiArea
API URL	https://tourism.api.opendatahub.bz.it/v1/SkiArea

it.bz.opendatahub.snowreport

The Snow Report Data dataset contains detailed reports of all South Tyrolean Ski Areas, including an aggregation of different Datatypes. Status and conditions of Lifts, Slopes, Ski tracks, Sledges, Measuring points (like e.g., snow height and last snow day), and Ski Areas basic data are part of this dataset.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Weather/get_v1_Weather_SnowReport
API URL	https://tourism.api.opendatahub.bz.it/v1/Weather/SnowReportBase
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Sources	LTS, IDM

it.bz.opendatahub.venue

This dataset contains **events location** provided by LTS such as info about a location (e.g. number of seats, number of seats for disabled people, venue, etc.) or info to book an event location (e.g. prices, room configuration, etc.). Data are accessible in AlpineBits DestinationData format.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/Venue
API URL	https://tourism.api.opendatahub.bz.it/v1/Venue
Use cases and info	https://databrowser.opendatahub.bz.it/
Sources	LTS

it.bz.opendatahub.weather-forecast

This dataset contains weather forecast for South Tyrol. Updated daily at 07:00 AM and 11:00 AM (Sundays only at 10 AM), the bulletin contains the following forecasts:

- current and next day weather, and five days evolution in the whole South Tyrol
- for current and next day in mountain
- current weather and next two days evolution for each South Tyrolean districts.

All data is requested live through the Weather service of the Province of Bolzano-Bozen (South Tyrol). Data is available in English, German, and Italian language.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/weather
Use cases and info	https://databrowser.opendatahub.bz.it/
Android App	Südtirol Guide/Alto Adige Guide
Sources	SIAG

it.bz.opendatahub.weather-history

This dataset is a collection of ten years of data, collected from 2010 onwards, about the meteorological provided by the *Weather and Avalanche Services* of South Tyrol and Trentino.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
StationType	WeatherHistory
Use cases and info	https://databrowser.opendatahub.bz.it/
Sources	Weather and Avalanche Services of South Tyrol and Trentino

it.bz.opendatahub.webcam

This dataset contains Webcam Data. All data is synchronized daily from LTS (Webcam links). All data is transferred and is daily updated.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v1
Swagger URL	https://tourism.api.opendatahub.bz.it/#/WebcamInfo
API URL	https://tourism.api.opendatahub.bz.it/v1/WebcamInfo
Use cases and info	https://databrowser.opendatahub.bz.it/ https://suedtirol.info/en/
Sources	Webcam links from LTS and IDM

3.5 Datasets in Other Domains

Creative Industries

This dataset originates from data collected in the **DAVINCI** project, whose aims is to increase local businesses by improving innovation and digitalisation of services.

The data consists of a list of actors that operated in the **creativity sector** in South Tyrol and is used as data source to generate a map for a dedicated *Web Component*

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v2
StationType	CreativeIndustry
Web Component	Creative Industries
Sources	IDM

NOI-Place

This dataset contains all places, offices, meeting rooms, and in general all points of interest located in the various buildings that compose the NOI Techpark in Bolzano-Bozen, Italy.

This dataset is used to create the interactive map of the NOI Techpark.

Output	JSON, mime-type application/json
E-mail contact	help@opendatahub.bz.it
API version	v2
StationType	NOI-Place
Use cases and info	https://maps.noi.bz.it/
Sources	NOI Techpark

<https://alpinebits.opendatahub.bz.it/AlpineBits>

This endpoint provides data from the **AlpineBits Open Standard**, whose purpose it to ease B2B data exchange among actors involved in the alpine tourism.

Documentation for developers, including sample code snippets and additional material can be found on AlpineBits dedicated page for developers <https://www.alpinebits.org/hoteldata/>.

Various development and testing tools can be found on <https://development.alpinebits.org/#/home>.

Output	XML, multipart/form-data
E-mail contact	help@opendatahub.bz.it
AlpineBits versions	v2017-10, v2018-10 (coming soon 2020-10)
AlpineBits Endpoint	https://alpinebits.opendatahub.bz.it/AlpineBits

LIST OF HOWTOS

This page contains the list of available howtos, divided into areas. The list of howtos, together with a short description is available here:

4.1 Mobility

1. *How to Access Mobility Data With API v2* Getting started with the new API v2.
2. *How to Access Analytics Data in the Mobility Domain* This howto guides you in browse and query data produces from the sensors used in the mobility domain.
3. *How to access e-Charging Stations Data?* This howto is deprecated and has been removed, please refer to *How to Access Mobility Data With API v2* if you are interested in accessing the mobility dataset.

4.1.1 How to Access Mobility Data With API v2

The new **API v2** (see [the description](#)) for the Mobility domain has simplified the access to data; among its features, we recall that there is now one single endpoint from which to retrieve data from all datasets.

The starting point for all actions to be carried out on the datasets made available by the Open Data Hub team is the swagger mobility API home page:

<https://mobility.api.opendatahub.bz.it/>

Note: This page contains also the deprecated Mobility V1 API, for backward compatibility only. This API may be removed in the future, so please do not rely on it and use the API V2 **only**.

From this site, links provide access to documentation about data licencing and use of the API; it is also possible to contact the Open Data Hub team by sending an email to the issue tracker, to ask questions,

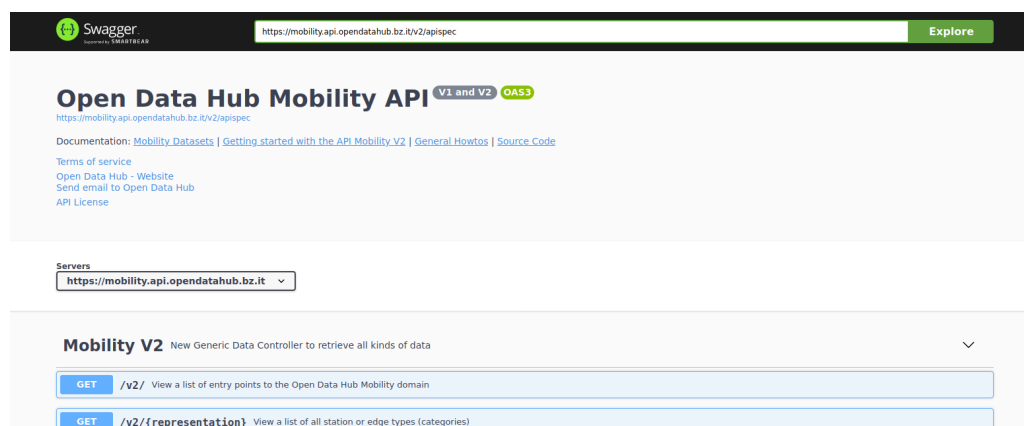


Figure 4.1: The swagger interface of the Mobility API v2.

provide feedback,
or to report issues.

When executing queries against the datasets, besides the output data, you will always receive the *curl* command and the corresponding direct URL, to be used for examples, in scripts. See [Example Queries](#)

Getting Started

In the API v2, the central concept is **Station**: all data come from a given *StationType*, whose complete list can be retrieved by simply opening the second method of the **Mobility V2** controller, */v2/{representation}*, then click on Try it out and then on *Execute*.

Station types in the resulting list can be used in the other methods to retrieve additional data about each of them. To check which station belongs to which datasets, you can check the list of [Datasets in the Mobility Domain](#).

Example Queries

We use some of the techniques presented in section [Advanced Data Processing](#) and the [parking dataset](#) to show a few simple queries and see the output they provide.

We recall that the two *StationTypes* of that datasets are *ParkingStation* and *ParkingSensor*; we start by retrieving all *ParkingStations*, leaving all other options to their default values, and then building two more refined queries, one using the select clause and one using the where clause:

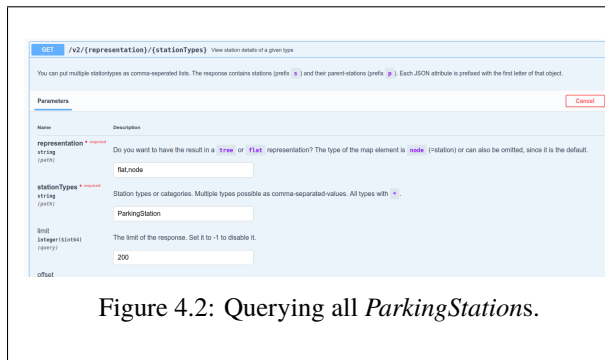


Figure 4.2: Querying all *ParkingStations*.

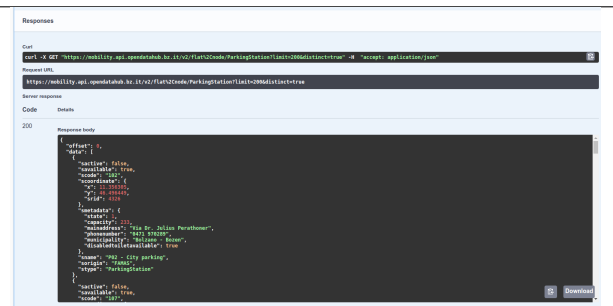
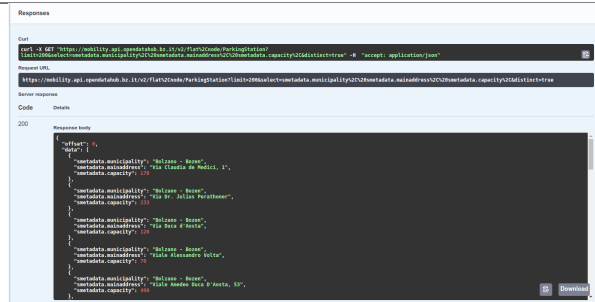
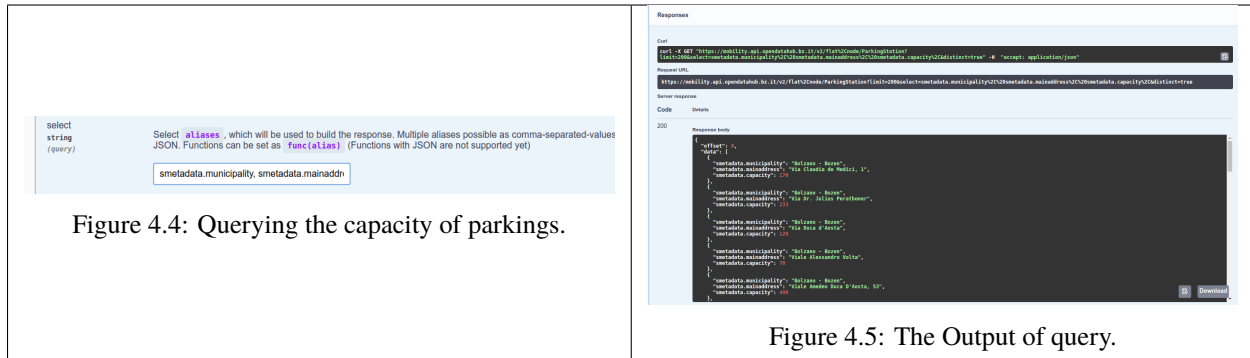


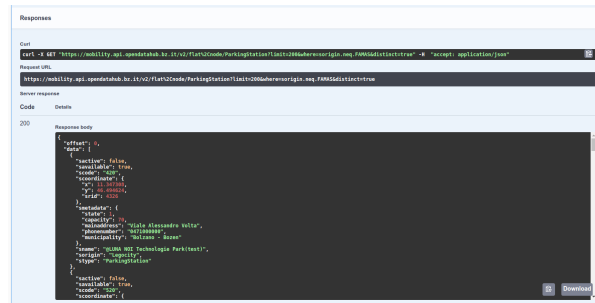
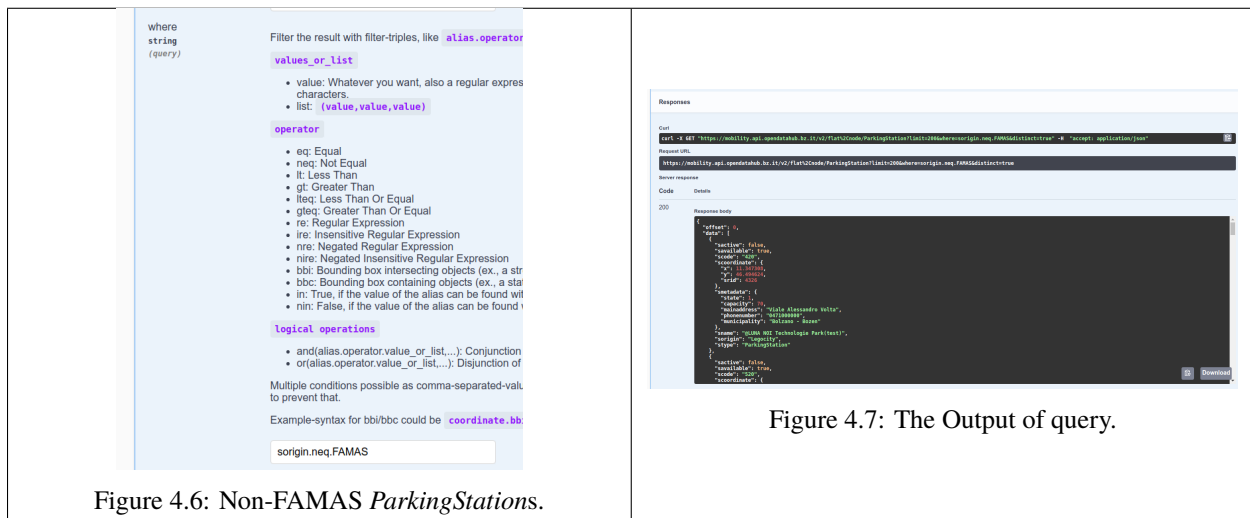
Figure 4.3: The Output of query.

Hint: You can copy to the clipboard or download the result of the query by clicking on the bottom-right corner icons.

If we would like to know the capacity of each of the parking lots, we add *smetadata.municipality*, *smetadata.mainaddress*, *smetadata.capacity* to the **select** clause:



Finally, we are interested only in the *ParkingStations* whose origin is **not** FAMAS. We need therefore to add the following to the **where** clause (we also remove the entry added for the previous query in the **select** clause):



You can build more complex queries by simply adding more entries to the Select and where clauses.

4.1.2 How to Access Analytics Data in the Mobility Domain

This howto guides you in browsing and querying data from the Mobility domain using the <https://analytics.opendatahub.bz.it/> web site.

Note: Access to data on this website is now possible by using R. See [the dedicated section](#) for information and directions.

Introduction

The website <https://analytics.opendatahub.bz.it/> gathers data from datasets in the mobility domain and uses them to draw two types of diagram: a *chart* using historical data and an interactive *map* that show where are located the sensors on the territory. The latter is the default landing page.

Charts

The charts page contains a number of options to show data, both historical and current, from the mobility domain. Data are gathered by sensors which are installed on various locations in South Tyrol and are operated and governed by different institutions or public and private companies. While the vast majority of the data comes from South Tyrol, there are datasets (including the E-mobility, weather, and traffic domain), which contain data about some neighbouring Italian provinces and regions.

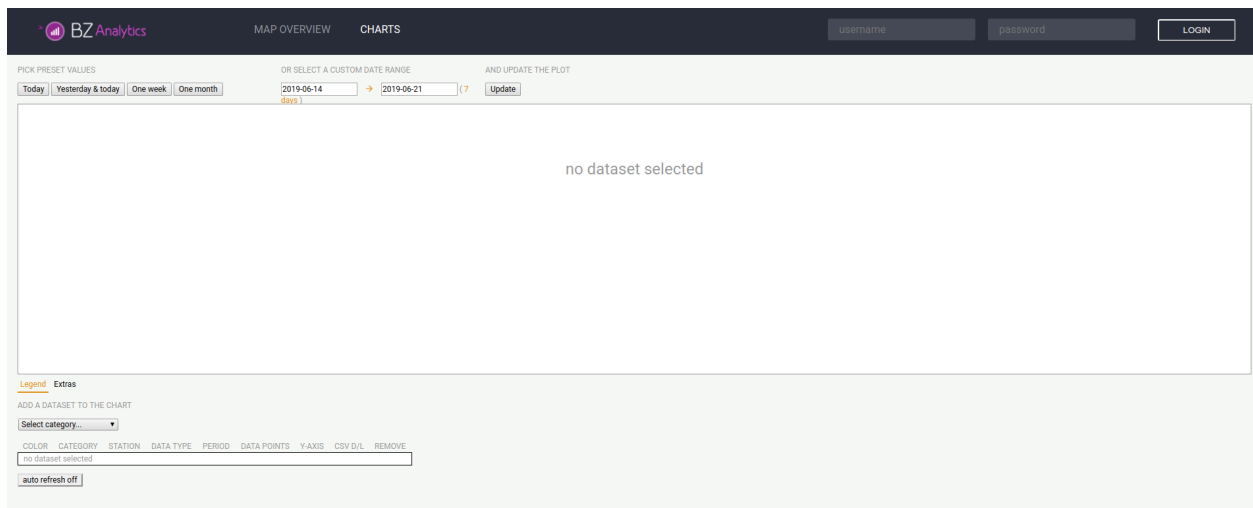


Figure 4.8: The landing page of analytics.mobility.bz.it.

While there are many controllers in the page, that allow to tweak the search parameter, the basic usage is quite simple and requires two steps:

1. Select a dataset to be added to the chart, from the drop-down menus below the diagram.
2. Restrict the data to be displayed to a date range, either a predefined one or a custom one.

A sample display from the weather datasets is shown in [Figure 4.9](#), in which data from only one temperature sensor are used, and in [Figure 4.9](#), using data from two temperature sensors.

Map Overview

In the map overview, there is a map, initially displaying only the South Tyrol region, with the list of available sensor types on the left-hand side. When clicking on one or more items, the position of all sensors will appear on the map, see [Figure 4.11](#) for the parking lots available in the Trentino-South Tyrol region.

A signpost with a circled + indicates that there are more sensors around at that location; this is true especially when the map encompasses a large area, like e.g., the whole South Tyrol region. Therefore, by zooming in on the map, or by (repeatedly) clicking on the +, more signposts will appear, until the + either disappears or is replaced by a different sign: you have found the (unique) sensor at that location.

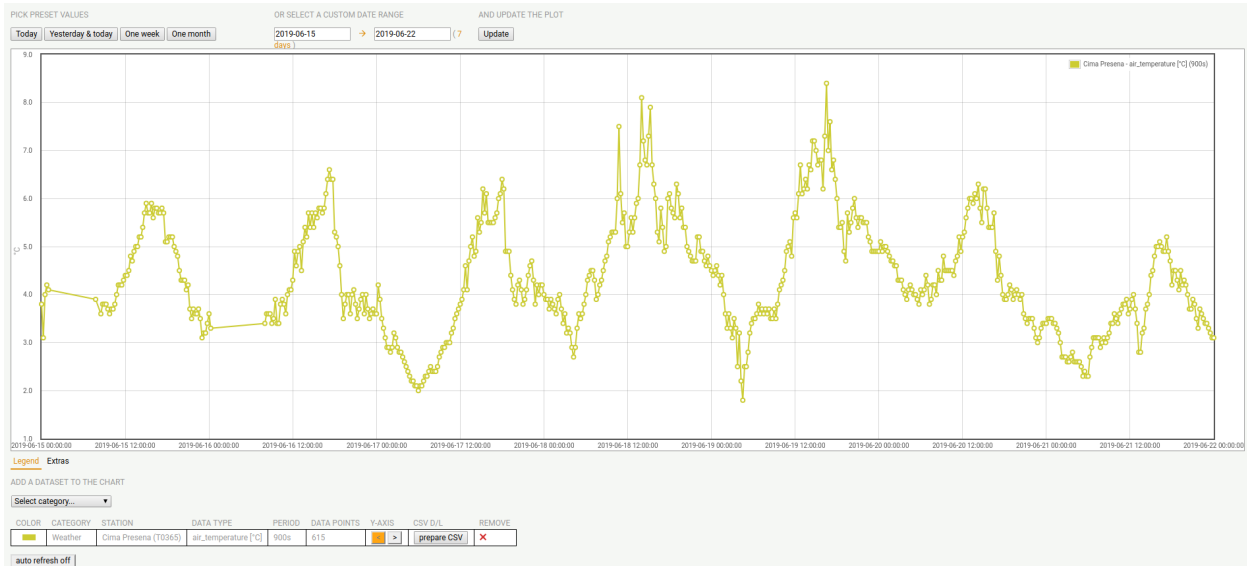


Figure 4.9: Sample temperature diagram on Cima Presena.



Figure 4.10: Sample temperature diagram on Cima Presena and Cima Paganella.

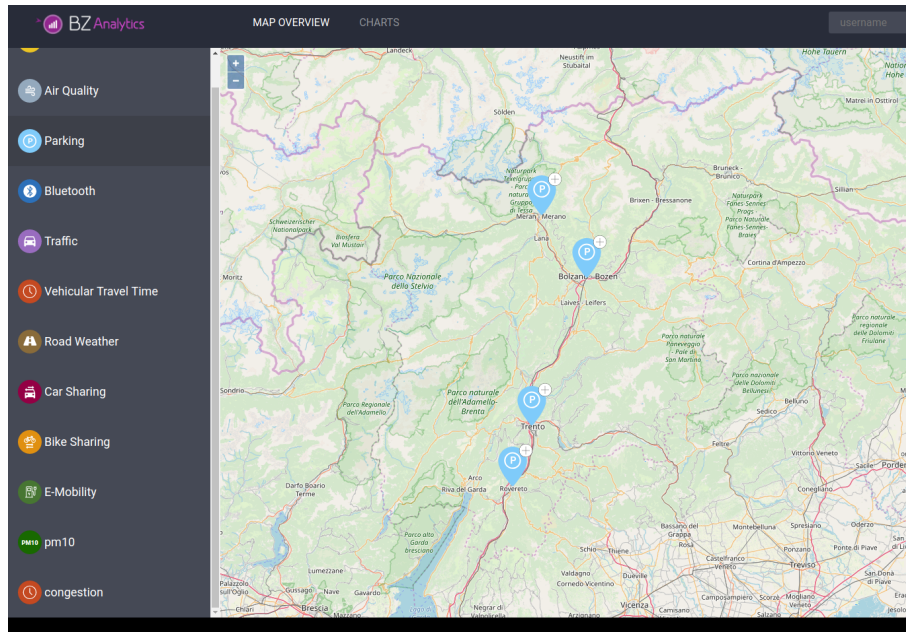


Figure 4.11: Map with parking lot signposts.

In the case of Parking data—and in a few other datasets, the + will be replaced by a green, yellow, or red circle, meaning that there are many, a few, or no free parkings in that lot.

For other types of sensors, the + simply disappears.

When clicking on a single sensors, a panel will appear on the right-hand side, containing a lot of information about that sensor, including its unique ID within the dataset, geographic coordinates. Additional information displayed depend on the dataset.

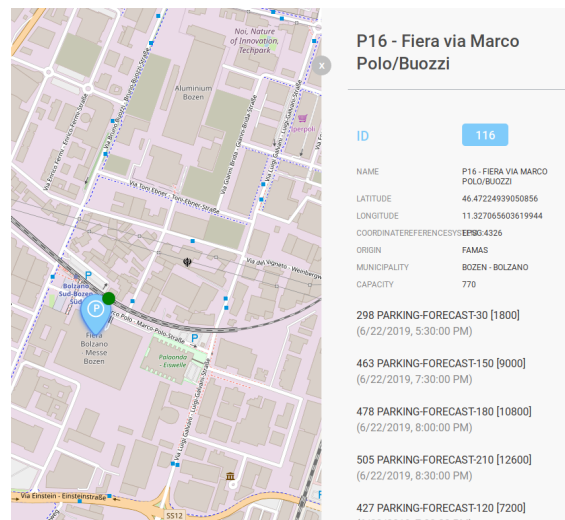


Figure 4.12: Details of a sensor.

4.2 Tourism

1. *How to access Tourism Data?* Description of how to access and manipulate data, the input data used, and the various filters available in the Tourism domain.
2. *How to use the Open Data Hub's Tourism Data Browser?* Access to the open data provided within the Tourism domain.
3. *Quick and (not-so) Dirty Tips for Tourism (AKA Mini-howtos)* Mini howtos, tricks&tips, and use cases for data in the Tourism domain.
4. *How to access Open Data Hub AlpineBits Server as a client* access AlpineBits data

4.2.1 How to access Tourism Data?

The purpose of this howto is to quickly introduce the alternatives to access datasets in the Tourism domain. Technical information about the datasets can now be found in section *Technical Information for Tourism Dataset*.

Swagger Interface

All the APIs available for the tourism domain can be accessed from the same URL through their Swagger user interface:

<https://tourism.api.opendatahub.bz.it/swagger/index.html>

Hint: Check section *Datasets in the Tourism Domain* for direct URLs to the datasets.

With the introduction on the Tourism API graphic interface of a newer swagger version, supplying and storing the token has become easier, making older procedures deprecated or obsolete. Moreover, in the new GUI, for every API method is shown whether it can provide Open Data as a result and if not, it will be necessary to authenticate.

Authentication with Swagger

Authentication is easy and, unlike it happened in the past, it does require only to supply your credentials. From the swagger UI, click on the **Authorize** button on the right-hand side of the page (shown in the bottom-right corner in Figure 4.13).



Figure 4.13: The new Swagger UI for Tourism domain.

A dialog window will pop up; here, supply your username and password, and click on **Authorize**. It is not necessary to change any other parameter.

After a few seconds a new dialog replaces the one used for authentication, whose most important bit is the **Authorized** word, that means you are now authenticated. No additional step is now necessary: the browser will remember the token. Click on **Close** to close the dialog window start browsing the Tourism data.



Figure 4.14: Providing credentials for authentication.



Figure 4.15: Successful authentication.

To log out, click again on **Authorize** in the Swagger UI (see [Figure 4.13](#)), then on **Logout**.

Browsing API Datasets

The data in the API can be browsed at the following URL:

<https://tourism.api.opendatahub.bz.it/v1/>

Note: You may need to install a *JSON browser plugin* for your browser to browse the datasets in this way.

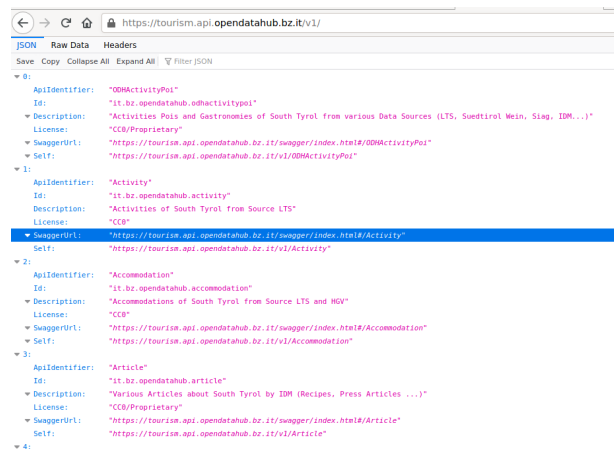


Figure 4.16: Browsing Tourism API

Here, every link can be clicked to navigate through the various datasets and the data they contain. A lot of metadata and information is provided for every object in the dataset, depending on the type of object. For example, The starting point to browse the API, shown in [Figure 4.16](#) includes for each datas licensing information, a description, an ID, the API and swagger URLs, while a dataset shows the total number of items and of pages it contains, the current page, pointers to previous and next page, and the items themselves.

Using Command Line Tools

If you plan to access the API methods with command line tools like **curl** or **wget**, or only from scripts, you need to add an authentication header to each call. For example, using curl:

```
~# curl -X GET --header 'Accept: application/json' \
--header 'Authorization: Bearer vLwemAqrLKVKXsvgvEQgtkeanbMq7Xcs' \
'https://tourism.api.opendatahub.bz.it/v1/Gastronomy'
```

Note: The string of the token is shortened for the sake of clarity.

It is important to mention that the authorisation header requires the following syntax: **Authorization: Bearer**, followed by the whole *string* of the token.

Once you have retrieved the data, which come in JSON format, you can process and manipulate them with a tool like jq.

4.2.2 How to use the Open Data Hub's Tourism Data Browser?

This how-to explains the necessary steps to access and retrieve data from the Open Data Hub's tourism domain.

Data Browsing and Exploring

In order to access the data in the tourism domain, launch a browser and point it to <https://databrowser.opendatahub.bz.it/>.

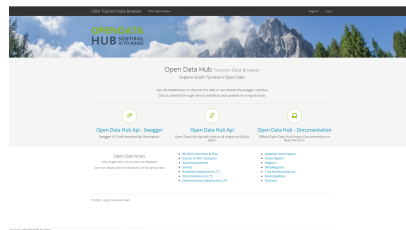


Figure 4.17: The home page of the Tourism Data Browser.

Under the header and an informative message, hyperlinks in the centre of the page allow to open various resources about the Tourism domain.

- The [Swagger interface](#) of the datasets, the quickest place from where to access the datasets and learn how to programmatically retrieve the data
- The [Open Data Hub API](#), a browsable, hyperlinked interface over the data in the Tourism domain.

Note: Depending on the browser used to access the APIs, an extension might be needed for the JSON to be properly displayed and browsed. For example, on Chrome you should install <https://github.com/callumlocke/json-formatter>.

- The [Official documentation](#) of the Open Data Hub project

The bar at the top of the page allows to carry out a few actions:

- **ODH Open Data.** This drop-down menu allows to choose the dataset from which to browse the data. These can be reached also using the hyperlinks in the lower part of the home page.

- **Register.** This is currently not active, and redirects to *Log in*.
- **Log in.** Allows to access the data browser as a registered user, for example to add or edit some data. If you have access credentials, write the username (e-mail address) and password that were provided to you and click on Log in. You will be redirected to the home page as a logged in user and from here, you will see the box with the permissions you have to access the various datasets and be able to modify data.

When you access the **ODH Data** item in the top menu, you will be able to select a dataset among those available. As an example, [Figure 4.18](#) shows what is available in the *ODH Open Data* → *Activities & Pois* → *Winter* filter - in this case a list of activities that can be done during the winter on the snow.

The page allows to further filter the results, by using search strings and/or the list of tags underneath, to move between pages of results, and to change language of the interface (although at the moment the page is not fully translated in all languages!)

The screenshot shows the 'ODH Tourism Data Browser' interface. At the top, there's a navigation bar with 'ODH Open Data' and 'Register Log in' links. Below it, the title 'Activities & Pois - Winter' is displayed. A language selector shows 'de', 'it', 'en', 'nl', 'cs', 'pl', 'fr', 'ru'. Below the language selector, there's a pagination bar showing 'Elements: 2023', 'prev', '1 / 102', and 'next'. On the left, there's a filter sidebar with sections for 'Activity/Poi Name', 'Location', 'ODH Tag', 'Highlight', and 'Type'. The 'Type' section is expanded, showing a list of winter activities: 'Weihnachtsmärkte', 'Eisklettern', 'Eislaufen', 'Langlaufen', and 'Pferdeschlittenfahrten'. The main content area displays a table of activities with columns: 'Main image', 'Shortname', 'Type', 'Location', 'Languages', 'Source', 'OA', 'Active', 'ODH active', and a 'Detail' button. The table lists five activities, all of which are 'Winter - Cross-country skiing' at 'Three Peaks Dolomites - Dobbiaco/Toblach'.

Main image	Shortname	Type	Location	Languages	Source	OA	Active	ODH active	
	Magic Town - The Val Gardena Christmasmarket for children and families!	Winter - Christmas & Markets - Rural Christmas Markets	Val Gardena/Gröden - S. Cristina/St. Christina	de, it, en	Magnolia		✓	✓	Detail
	01 Cross Country Stadlo Track Dobbiaco/Toblach	Winter - Cross-country skiing	Three Peaks Dolomites - Dobbiaco/Toblach	de, it, en	LTS	OA	✓	✗	Detail
	02 Cross Country Stadlo Dobbiaco/Toblach FISU - Stephanie	Winter - Cross-country skiing	Three Peaks Dolomites - Dobbiaco/Toblach	de, it, en	LTS	OA	✓	✗	Detail
	03 Cross Country Stadlo Dobbiaco/Toblach FISU - Ole	Winter - Cross-country skiing	Three Peaks Dolomites - Dobbiaco/Toblach	de, it, en	LTS	OA	✓	✗	Detail
	04 Cross Country Stadlo Dobbiaco/Toblach FISU - Nuthalle	Winter - Cross-country skiing	Three Peaks Dolomites - Dobbiaco/Toblach	de, it, en	LTS	OA	✓	✗	Detail
	05 Cross Country Stadlo Dobbiaco/Toblach FISU - Kunka	Winter - Cross-country skiing	Three Peaks Dolomites - Dobbiaco/Toblach	de, it, en	LTS	OA	✓	✗	Detail

Figure 4.18: Accessing the data through filters or menu item.

If you click on the image associated to each item in the list or on the **Detail** button, an overlay will pop up, which contains more detailed information about that activity.

Note: Images in the list are displayed only if they are uploaded with a CC0 license.

Logged in Users

When you access the Tourism Data Browser using credentials that have been provided to you by the Open Data Hub team, the appearance of the page slightly changes

In particular, at the bottom of the page a table with the user's role and permissions replaces the list of datasets, and an additional menu item (**external Data Sources**) appears in the top bar, allowing access to some more datasets.

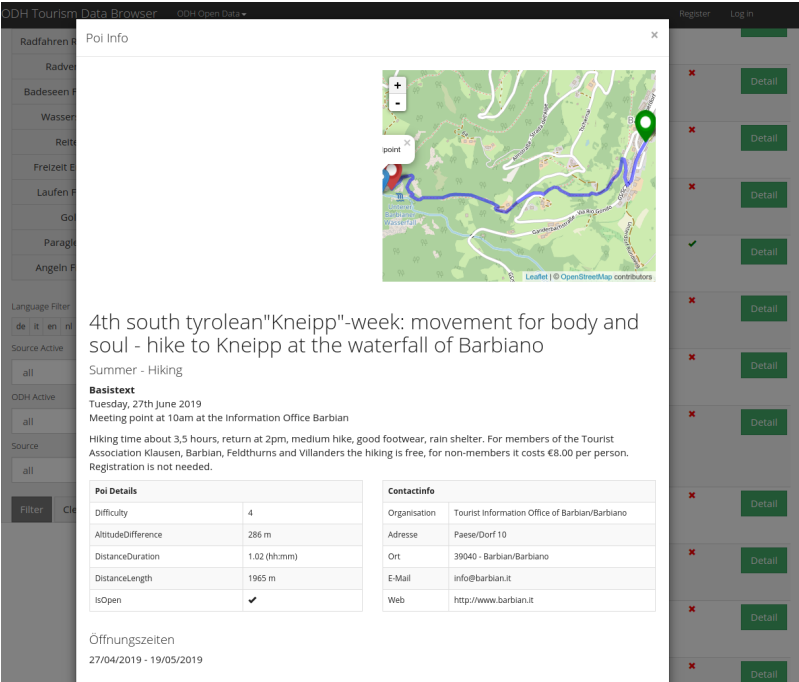


Figure 4.19: Detailed view of a POI (Point Of Interest).

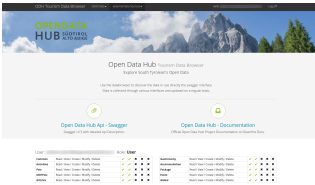


Figure 4.20: The home page after a successful login.

4.2.3 Quick and (not-so) Dirty Tips for Tourism (AKA Mini-howtos)

This section contains various tips and tricks to improve and tweak the queries sent to the Tourism datasets, allowing more precise results to be retrieved. This page is divided into two parts: The first one shows examples with code (usually the API call), the second is organised like a FAQ section.

Example Calls

EX1. Why does this query return no result?

<https://tourism.api.opendatahub.bz.it/v1Gastronomy?pagesize=3&categorycodefilter=0&locfilter=reg268>

Because there is no value **reg268** for *locfilter*. You can return valid IDs to be used as *locfilter* using this call:

<https://tourism.api.opendatahub.bz.it/v1RegionReduced?language=it>

An example result for this call is:

```
{
  "Id": "D2633A26C24E11D18F1B006097B8970B",
  "Name": "Alta Badia"
}
```

Therefore, use the ID **regD2633A26C24E11D18F1B006097B8970B** in *locfilter* to search for Gastronomy in the Alta Badia region.

EX2. The *locfilter* parameter.

Q: How do I correctly use the *locfilter* parameter?

```
locfilter =>
Locfilter (Separator ',', possible values: reg + REGIONID = (Filter by
Region), reg + REGIONID = (Filter by Region), tvs + TOURISMVEREINID =
(Filter by Tourismverein), mun + MUNICIPALITYID = (Filter by
Municipality), fra + FRACTIONID = (Filter by Fraction)),
(default: 'null')
```

It seems to accept a string, but how is this string built?

A: *locfilter* accepts a string composed as follows: a region identifier, followed immediately by a location Identifier.

Location identifier are the following four:

- **reg:** Region (Italian Regione)
- **tvs:** Turistic association (German Tourismusverein)
- **mun:** Municipality, i.e., town or city (Italian Municipalità)
- **fra:** Suburb or district (Italian frazione)

IDs for each location can be gathered either from the swagger interface or using an API calls:

- **reg:**
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_GetRegionsReduced
<https://tourism.api.opendatahub.bz.it/v1RegionReduced?language=it>

- **tvs:**

```
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_GetTourismvereinReduced
https://tourism.api.opendatahub.bz.it/v1TourismAssociationReduced?language=iturismusverein)
```

- **mun:**

```
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_GetMunicipalityReduced
https://tourism.api.opendatahub.bz.it/v1MunicipalityReduced?language=it
```

- **fra:**

```
http://tourism.opendatahub.bz.it/swagger/ui/index#!/Common/Common_GetDistrictReduced
https://tourism.api.opendatahub.bz.it/v1DistrictReduced?language=it
```

For example, to retrieve all Gastronomy in the suburb of Lana, first retrieve its ID, which is:

```
{
  "Id": "79CBD79551C911D18F1400A02427D15E",
  "Name": "Lana"
}
```

Then pass the string **fra79CBD79551C911D18F1400A02427D15E** as *locfilter*:

```
https://tourism.api.opendatahub.bz.it/v1Gastronomy?locfilter=fra79CBD79551C911D18F1400A02427D15E
```

EX3. The *categorycodefilter* parameter.

Q: *categorycodefilter* seems similar to the *locfilter* parameter found in [this trick](#), but this does not accept string?

```
Category Code Filter (BITMASK values: 1 = (Restaurant), 2 = (Bar /
Café / Bistro), 4 = (Pub / Disco), 8 = (Apres Ski), 16 =
(Jausenstation), 32 = (Pizzeria), 64 = (Bäuerlicher Schankbetrieb),
128 = (Buschenschank), 256 = (Hofschank), 512 = (Törggele Lokale),
1024 = (Schnellimbiss), 2048 = (Mensa), 4096 = (Vinothek /Weinhaus /
Taverne), 8192 = (Eisdiele), 16384 = (Gasthaus), 32768 = (Gasthof),
65536 = (Braugarten), 131072 = (Schutzhütte), 262144 = (Alm), 524288 =
(Skihütte)
```

The *categorycodefilter* parameter accepts integers instead of strings, in *bitmask-value*. The code of each category is a power of 2, so to search in multiple categories, simply **add** the respective codes and pass them as value of the parameter. For example, to search for Restaurants (1) and Pizzerias (32), pass **33** to *categorycodefilter*:

```
https://tourism.api.opendatahub.bz.it/v1Gastronomy?categorycodefilter=33
```

Tips and Tricks

TT1. Categorycodefilter in the Accommodation dataset.

Q: In the Accommodation dataset there's no *categorycodefilter* filter, like in the Gastronomy dataset. Is there some equivalent filter?

A: In the Accommodations dataset use **categoryfilter** instead.

TT2. *odhactive* and filters starting with *odh*.

Q: What is the purpose of the *odhactive* filter? And what do all the filters prefixed with **odh** stand for?

A: In the datasets, there are filters like *active* and *odhactive*, where *odh* simply stands for Open Data Hub. Filters starting with **odh** are collectively called *ODHtags*.

Datasets filtered with the former return all data sent by the dataset provider, while the latter returns those validated by the Open Data Hub team as well. This parameter is useful in a number of use cases. Suppose that the Open Data Hub team receives a dataset contains name and location of ski lifts within South Tyrol's ski areas. If the dataset has not been updated in a few years, some entry in that dataset might be non valid anymore, for example a ski lift has been replaced by a cable car or has been dismantled. If this case has been verified by the Open Data Hub team, the entry referring to that ski lift will not appear in the Open Data Hub.

TT3. The *seed* filter

Q: What is the *seed* filter used for?

A: *seed* is used in pagination, i.e., when there are two or more pages of results, to keep the sorting across all pages. When retrieving a high number of items in a dataset it is desirable to have only a limited amount of results in each page.

It is possible to activate seed in two ways: in the dataset, choose a *pagenumber* (the number of the result page that will be shown first) or a *pagesize* (number of items in each page, we'll use **15** in this example) and set seed to **0**. At the beginning of query's **Response Body** you will see something like:

```
{
  "TotalResults": 10564,
  "TotalPages": 705,
  "CurrentPage": 1,
  "OnlineResults": -1,
  "Seed": "43",
  "Items": [
    {
```

The remainder of the **Response Body** contains the first 15 sorted items. If you now want to retrieve page 2, page 56, or any other, use **43** as seed and write **2**, **56**, or the desired value as *pagenumber*.

If you do not enter the **seed**, you could find an item that was already shown before, because the API can not guarantee that the same sorting is used in different queries.

4.2.4 How to access Open Data Hub AlpineBits Server as a client

In this howto, we show how to retrieve data from the AlpineBits Server endpoint for the Open Data Hub located at <https://alpinebits.opendatahub.bz.it/AlpineBits>, by using the (Linux) command line—in particular the **curl** application, and the popular *Postman* API development environment. An example call can be seen [at the bottom](#) of this section.

Note: Alternative command line programs like **wget** can be used as well: simply adapt the parameters described in the remainder of this section.

The first important thing to mention is that requests to this endpoint need a **POST** method and an authentication token, therefore you should specify options `--request POST` and a header requiring *Basic* authorization and containing your token: `--header 'Authorization: Basic <your-email-address>'`

Note: While there is no authorization required to access the Open Data Hub AlpineBits Server, we strongly suggest you to insert as token your email address for debugging reasons. It will help trace your calls in the case you require support from the Open Data Hub Team.

Next, it is necessary to provide the correct *client protocol version* and a *client ID*, which are two additional headers, namely `--header 'X-AlpineBits-ClientProtocolVersion: 2017-10'` and `--header 'X-AlpineBits-ClientID: 'My test request''`.

Concerning the client protocol version, it must be one of the supported version mentioned in [Table 4.1](#), which also shows the actions that can be used together with the protocol.

Table 4.1: Matrix of the protocol versions and supported actions in the AlpineBits implementation of the Open Data Hub.

Open Data Hub AlpineBits Server Actions	2017-10 PUSH	2017-10 PULL	2018-10 PULL	2018-10 PUSH
FreeRooms	Yes	No	Yes	No
GuestRequests	–	–	–	–
Inventory Basic	Yes	Yes	Yes	Yes
Inventory HotelInfo	Yes	Yes	Yes	Yes
RatePlans	–	–	–	–
BaseRates	–	–	–	–

Note that PUSH and PULL refer to the action of uploading to and downloading from AlpineBits Server, respectively.

Finally, to retrieve data from the AlpineBits Server, you need to set the correct content type (i.e., *multipart/form-data*) and provide an *action*. The content type is specified in another header by `--header 'Content-Type: multipart/form-data'`, while the action is given as a form: `--form 'action=getVersion'`.

Summing up what was described above, a call to the AlpineBits Server endpoint looks like the following one:

```
~# curl --location --request POST \
'https://alpinebits.opendatahub.bz.it/AlpineBits' \
--header 'Authorization: Basic <your-token-here>' \
--header 'X-AlpineBits-ClientProtocolVersion: 2017-10' \
--header 'X-AlpineBits-ClientID: 'My test request' \
--header 'Content-Type: multipart/form-data' \
--form 'action=getVersion'
```

Here, you can see that the additional option `--location` is used, that will make sure to resend the request in case a **3xx** HTTP error code is received, i.e., if the requested resource has been moved.

Postman Setup

In Postman, you need to enter the same data as in the call shown in previous section. The Postman setup equivalent to that call is shown in the two screenshots.

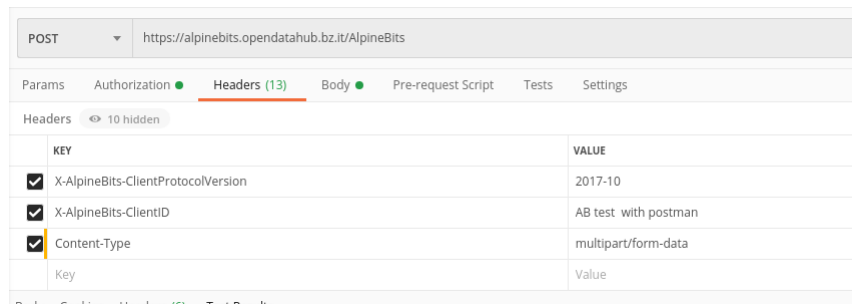


Figure 4.21: Definition of the call's headers.

In the headers you need to add all the parameters as in the curl version, except for the authentication: this option need to be specified in the *Authorization* tab of postman. Here, choose **Basic Auth** as type and use **someuser** and **secret** as username and password, respectively.

Note: It is suggested to use, instead of *someuser* and *secret*, your contact information, in order to be able to contact you for some technical reasons.

Next, you need to add, in Postman's *Body* tab, the action. Choose **form-data**, enter **action** as key and the name of the method to retrieve data, in our example **getVersion**.

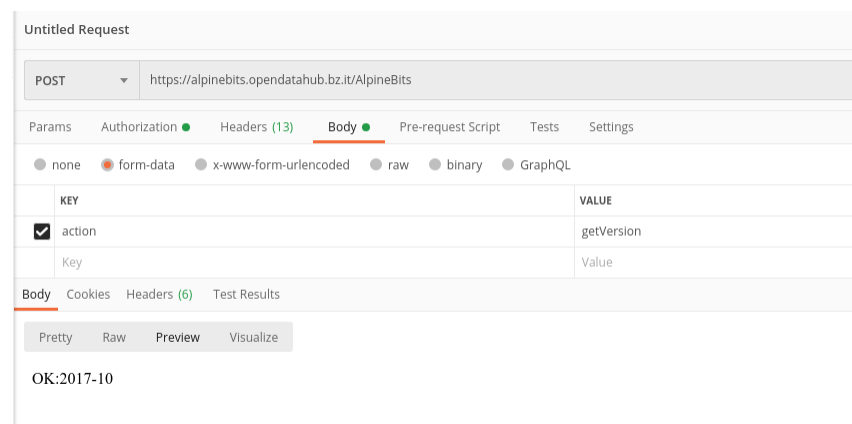


Figure 4.22: Definition of the *action* and the outcome of the call.

Once done, make sure to select **POST**, then click on Send and you will receive the result in the bottom part of Postman's window, like in Figure 4.22.

See also:

More information about the interaction with AlpineBits can be found in the official documentation, available at <https://www.alpinebits.org/hoteldata/>.

4.3 Generic HOWTOs

1. *How to use authentication?* Access to data that are not yet open (mostly for internal use).
2. *How to set up your local Development Environment?* Set up your workstation to develop for Open Data Hub—This tutorial is still in development and not so useful at the moment!
3. *How to set up Postman (API Development Environment)?* Setup of Postman, a popular API development environment, to access data from the Open Data Hub.
4. *How to insert and modify NOI Events?* Create and modify NOI events directly from the Open Data Hub portal.
5. *How to Publish your Web Component on the Open Data Hub Store* share your Web Components with Open Data Hub team
6. *How to Access Open Data Hub Data Using SPARQL* query Open Data Hub datasets using W3C's Query Language.
7. *How to Access Open Data Hub Data With R and SPARQL* query Open Data Hub datasets using R and its SPARQL library.

4.3.1 How to use authentication?

As described in section *Authentication in the Open Data Hub*, there are two methods to access protected data in the dataset: **Bearer Token Login** and **OAuth2 authentication**. Both authentication methods can be used within a browser or from the command line, with only slight differences.

Hint: The Open Data provided by the Open Data Hub can be freely accessed, authentication is meant only for internal use.


In this section we show how to use authentication within the Open Data Hub, provided that you own an username and a password to access the closed data in the datasets.

To obtain the credentials, please send an email to help@opendatahub.bz.it. This action will open a support ticket that will be taken in charge by the Customer Support Team.

Bearer Token Login

Bearer token login is used to access the *Datasets in the Tourism Domain*; description of the procedure is available at *Authentication with Swagger*.

OAuth2 authentication

OAuth2 authentication can be used in all the *Datasets in the Mobility Domain* that are marked with the  badge, so pick one dataset and go to its swagger interface, whose URL is provided together with the information of the dataset.

Note: As of Jun 30, 2023, authentication is not yet publicly available, so the following guidelines can not yet be put in practice.

If you use a browser

Make sure you have obtained a valid username and password, then open the `/rest/refresh-token` method and write you username and password in the two **user** and **pw** fields, respectively, as shown in [Figure 4.23](#).

The screenshot shows the OpenDataHub API interface for the `/rest/refresh-token` endpoint. The method is GET, and the description is "Request a new authorisation token to access protected data." Below the description, there is a note: "If you need to access protected, closed data and you have been given a username and password, invoke this method to receive a new token." The parameters section contains a table with two rows: **user** (required, string, query) with a description "The username of the user to which to grant the new token." and a value "JohnnyDoe"; and **pw** (required, string, query) with a description "The password corresponding to the user." and a value "my\$ecretPaw\$w0rd!". At the bottom of the parameters section is a blue "Execute" button. Below the parameters section is a "Responses" section.

Name	Description
user * required string (query)	The username of the user to which to grant the new token.
pw * required string (query)	The password corresponding to the user.

Figure 4.23: Request a new OAuth2 token.

If your credentials are valid, you will receive a new token, otherwise the response will be a **401 Unauthorized** error message.

The token you received can be used in any of the API's methods that require authorisation. A sample call is shown in [figure Figure 4.24](#). Note the syntax of the **Authorization** parameter: You must use prefix the authentication token with the **Bearer** string, followed by an empty space, then by the token.

In case you do not respect the `Authorization+space+token` sequence, use additional separators in the sequence (like [Figure 4.25](#) shows), or use an invalid token, you will receive an **401 - Unauthorized** HTTP response.

If you use the Command Line Interface.

Open a shell on your workstation and use a tool like **curl** or **wget**, with the appropriate options:

-X

Specify the request method (GET)

--header, -H

Add extra header information to be included in the request.

Note that the **--header** option is used twice: The first to receive the answer in **text/html** format, the second to provide the credentials required to access protected content.

API calls can be done using a tool like **curl** or **wget**, with the same **-X** and **--header** option used twice: The first to require the format of the response, the second to provide the credentials, like for example:

```
~$ curl -X GET "https://bdp-test-env.b7twguyhj.example.com/emobility/rest/get-records?
↪ station=83&name=CP1-Tignale&seconds=50" --header "Accept: */*" --header
↪ 'Authorization: Bearer <token>'
```

Make sure to replace the `<token>` with the actual token you received.

(header)

Bearer eyJhbGciOiJIUzUxMU9.eyJzdWIIOUJhbmFse

station * required
string
(query)
83

name * required
string
(query)
CP1-Tignale

seconds * required
integer
(query)
50

period
integer
(query)
period - period

Execute Clear

Responses Response content type */*

Curl

```
curl -X GET "http://bdp-test-env.b7twguhvj. /emobility/rest/get-records?station=83&name=CP1-Tignale&seconds=50" -H "accept: */*" -H "Authorization: Bearer eyJhbGciOiJIUzUxMU9.eyJzdWIIOUJhbmFseXRPY3NAaWRTLXN1ZWRoX3VjbC5jb20iLCJleHA1OjE1NjkwMzH1NDQsIn3VbGVzIjo1Uk9MRV9BTKFMVVRJQ1M1fQ.p7LZWRF4IPTc3IFvCDEFEU4H6B1t5SDjuk1b6m7K0IUsQ6H3U05m204whfWfWp409R00CZ_ykAb0tmsALQ"
```

Request URL

```
http://bdp-test-env.b7twguhvj. /emobility/rest/get-records?station=83&name=CP1-Tignale&seconds=50
```

Server response

Figure 4.24: A successful call to a method requiring authentication.

(header)

Bearer eyJhbGciOiJIUzUxMU9.eyJzdWIIOUJhbmFse

station * required
string
(query)
83

name * required
string
(query)
CP1-Tignale

seconds * required
integer
(query)
50

period
integer
(query)
period - period

Execute Clear

Responses Response content type */*

Curl

```
curl -X GET "http://bdp-test-env.b7twguhvj. /emobility/rest/get-records?station=83&name=CP1-Tignale&seconds=50" -H "accept: */*" -H "Authorization: Bearer eyJhbGciOiJIUzUxMU9.eyJzdWIIOUJhbmFseXRPY3NAaWRTLXN1ZWRoX3VjbC5jb20iLCJleHA1OjE1NjkwMzH1NDQsIn3VbGVzIjo1Uk9MRV9BTKFMVVRJQ1M1fQ.p7LZWRF4IPTc3IFvCDEFEU4H6B1t5SDjuk1b6m7K0IUsQ6H3U05m204whfWfWp409R00CZ_ykAb0tmsALQ"
```

Request URL

```
http://bdp-test-env.b7twguhvj. /emobility/rest/get-records?station=83&name=CP1-Tignale&seconds=50
```

Server response

Code	Details
401	Error: Unauthorized Response body { "status": 401, "name": "Unauthorized", "description": "ClientResponse has erroneous status code: 401 Unauthorized" }

Figure 4.25: A failed call to a method requiring authentication.

4.3.2 How to set up your local Development Environment?

This tutorial will guide you in the setup of the local infrastructure to be able to deploy, on top of the Open Data Hub, a new Data Collector Object starting from a simple **HelloWorld** template we provide.

This tutorial is divided into three parts:

1. Software installation
2. Services configuration
3. Troubleshooting

Software Installation

The following installation directions have been verified on a VM with installed either **Debian 9** or **Ubuntu 18.04.01 LTS**. The applications installed on it are the **Suggested** version.

Note: All the commands and configuration items (including their location in the filesystem) refer to this distribution and should be identical or quite similar on all other debian-based distributions as well.

On other Linux distribution some the name of the single packages might vary.

You need to install the following software:

Software	Minimum	Suggested	Notes
PostgreSQL	9.6	10.5	
postgis ext.	2.2	2.4	
Java	JRE7	JRE8	Most of the packages require Java 8 to be built.
git	2.17	2.17	
xmlstarlet	1.6.1	1.6.1	
Apache Maven	3.3.9	3.5.2	Optional. If you don't use it, do not install it.
tomcat8	8.0	8.5	You Optional can either use the tomcat server provided by Open Data Hub or install another application server.

Note: In case you opt to not use Maven or Tomcat, remember to edit the script in order to not attempt to configure them!

On a typical debian-based Linux distribution, installing the software is achieved by opening a shell/terminal, then issuing the following command, provided you have the rights to install software:

```
~$ sudo apt-get install git openjdk-8-jdk postgresql postgis maven tomcat8 xmlstarlet
```

This command ensures that all dependencies are installed as well. If you have none of these package already installed, you might need to download up to ~125Mb of packages.

Services configuration

The services will be configured automatically, since we developed a script that does most of the job for you. However, a few preliminary steps are required:

1. Make sure tomcat8 and postgres are running. If they do not or if you are unsure, refer to [entry 1](#) in section [Troubleshooting](#).
2. Verify that tomcat and postgres are listening on the right port (**8080** and **5432** respectively). See [entry 2](#) in section [Troubleshooting](#) for more information.
3. Make sure there is a database role configured with a password and a few access permission.
4. Set two environment variables.
5. Edit the script to suit your workstation.
6. Launch the script.

Warning: The script **might silently fail** on some machine, for example on Ubuntu 18.04, because it ships with Java 11. In this case, please install also java 8 and make it the default java version.

Troubleshooting

1. How do I check if a service is running?

You can check that a service like tomcat or postgres is running from the CLI, by issuing the following command and see an output similar to the one shown here, where the active (running) string can be read.

```
~$ service tomcat8 status
tomcat8.service - LSB: Start Tomcat.
   Loaded: loaded (/etc/init.d/tomcat8; bad; vendor preset: enabled)
   Active: active (running) since Wed 2018-06-13 16:36:28 CEST; 14min ago
     Docs: man:systemd-sysv-generator(8)
    CGroup: /system.slice/tomcat8.service
            └─13828 /usr/lib/jvm/java-8-openjdk-amd64/bin/java -Djava.util.logging.config.
               file=/var/lib/tomcat8/conf/lo

Jun 13 16:36:23 odh systemd[1]: Starting LSB: Start Tomcat....
Jun 13 16:36:23 odh tomcat8[13802]: * Starting Tomcat servlet engine tomcat8
Jun 13 16:36:28 odh tomcat8[13802]:    ...done.
Jun 13 16:36:28 odh systemd[1]: Started LSB: Start Tomcat..
```

If you do not use systemd, the command will have a different output:

```
~$ service tomcat8 status
[ ok ] Tomcat servlet engine is running with pid 11357.
```

From a browser you should connect to <https://localhost:8080/> (replace localhost with the URL or IP where your application server is located) and see the following page:

If tomcat is not running, start it using the following command, then entering your password.

```
~$ sudo service tomcat8 start
[sudo] password for odh:
```



Figure 4.26: The tomcat8 default landing page.

You can check again if tomcat is running with the command **service tomcat8 status**.

2. How do I check the port on which a service is listening?

You can use the **netstat** command line utility, like this:

```
~# netstat -plnt4
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/
↪ Program name
tcp        0      0 0.0.0.0:5432          0.0.0.0:*               LISTEN      2427/
↪ postgresql
tcp        0      0 0.0.0.0:22           0.0.0.0:*               LISTEN      2719/sshd
tcp        0      0 127.0.0.1:8080       0.0.0.0:*               LISTEN      2863/
↪ tomcat8
```

Make sure that at least ports 8080 and 5432 are present (tomcat and postgres respectively) in the **Local Address**.

It is suggested to run this command as superuser, because otherwise not all information is present.

4.3.3 How to set up Postman (API Development Environment)?

Postman is a popular API development environment, that is, a tool that is used (among other useful features) to ease the interaction with API calls to remote sites. In this tutorial, we show the few steps necessary to set Postman to connect to the Open Data Hub datasets in both the mobility and tourism domains.

In the remainder of this tutorial, we will use as example the *E-chargin station* dataset, located at <https://swagger.opendatahub.bz.it/?url=https://mobility.api.opendatahub.bz.it/v2/apispec#/Mobility%20V1%20-%20Emobility/> for the mobility domain and the *Accommodation* dataset, located at <https://tourism.api.opendatahub.bz.it/#Accommodation>.

Initial Setup

After Postman has been launched, click on the New button, then on Request to start the configuration of the Open Data Hub endpoints, like shown in Figure 4.27.

In the dialog window that opens, write the URL of the endpoint in the *Request name* textfield and assign it in the ODH collection, see Figure Figure 4.28.

Hint: If no collection has already been created, create one by clicking on + Create collection, then write **ODH** and confirm.

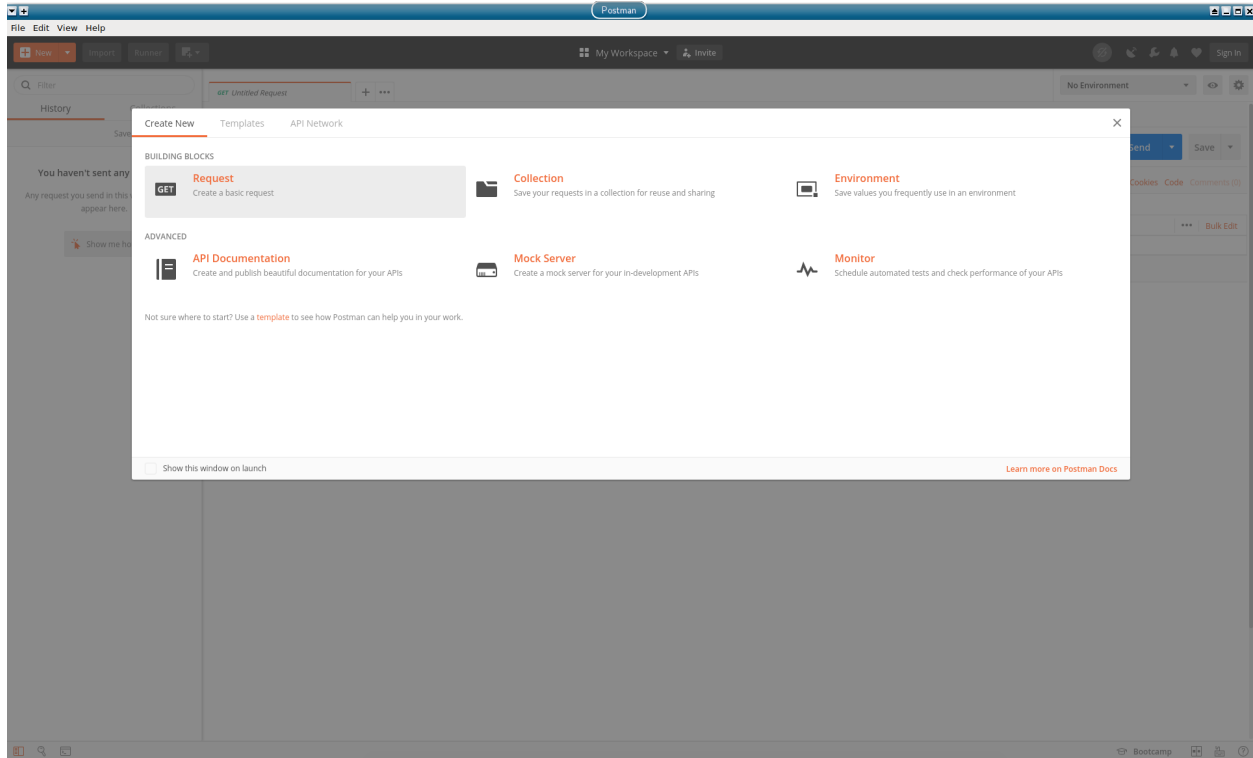


Figure 4.27: Start of a new request creation.

Click on Save to ODH to start querying the endpoint.

Repeat the procedure for the Accommodation dataset and for any other dataset you want to query.

It is now possible to start querying the endpoints, by providing next to the **GET** button the corresponding call, like shown in Figure 4.29 for the E-charging station dataset and in Figure 4.29 for the Accommodation dataset. However, while the former images shows a set of results, on the latter appears the message *Authorization has been denied for this request.* and the status 401 Unauthorized.

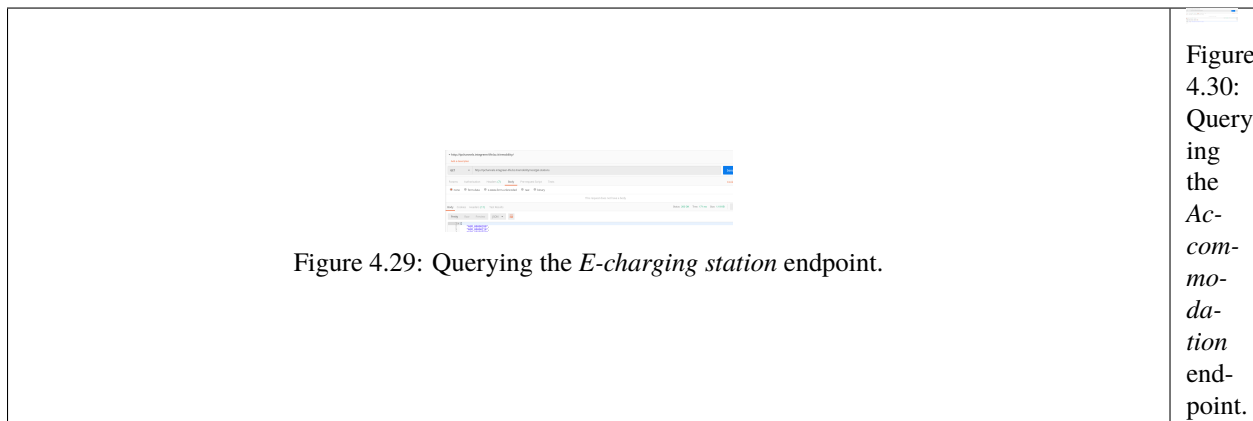
Figure 4.29: Querying the *E-charging station* endpoint.

Figure 4.30: Querying the Accommodation endpoint.

The reason is that the data contained in that dataset have not (yet) been published as open data, therefore authentication is necessary. This is where Postman proves useful, since it can request authentication tokens (OAuth2 in the case of Open Data Hub), store them, and use them whenever they are needed.

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
[Learn more about creating collections](#)

Request name
 http://ipchannels.integreen-life.bz.it/ermobility/swagger-ui.html

Request description (Optional)
 Adding a description makes your docs better

Descriptions support Markdown

Select a collection or folder to save to:

Search for a collection or folder

◀ ODH + Create Folder

Cancel Save to ODH

Figure 4.28: Defining a new endpoint in the mobility domain.

Getting a new Authorisation Token

To request a new authorisation token, click on *Authorization* right below the GET request, then select OAuth 2.0 as the *Type*.

Now, in the right-hand side of the window, write the URL that manages the tokens (for the tourism domain, this is <https://tourism.opendatahub.bz.it/token> and click on the Get New Access Token button (Figure 4.31).

https://tourism.opendatahub.bz.it/api/Accommodation

GET https://tourism.opendatahub.bz.it/api/Accommodation Send Save

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code Comments (0)

TYPE
 OAuth 2.0

The authorization data will be automatically generated when you send the request. [Learn more about authorization](#)

Add authorization data to
 Request Headers

Preview Request

Access Token
 https://tourism.opendatahub.bz.it/token Available Tokens

Get New Access Token

body Cookies Headers (10) Test Results Status: 404 Not Found Time: 267 ms Size: 414 B Download

Figure 4.31: Requesting an access token.

In the dialog window that opens fill in all the necessary fields, like shown in Figure 4.32, selecting **Password Credentials** as the *Grant Type*, then click on Request Token. Make sure you have received the username and password to obtain the token, and give it a name easy to remember.

If your credentials are correct and the request is successful, the dialog window will be replaced by another one containing the access token and a few details about it, including its validity and expire date, see Figure 4.33 and Figure 4.34.

Figure 4.32: A filled-in token request.

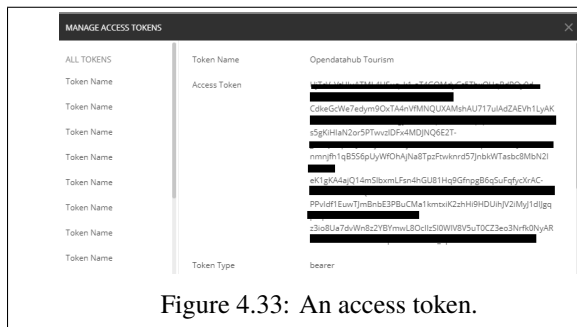


Figure 4.33: An access token.

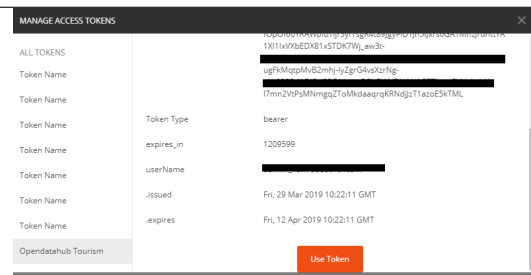


Figure 4.34: Information about an access token

It is now possible to select the token: Select **Opendatahub Tourism** from the *Available Tokens* drop-down menu (see Figure 4.31), click on *Body* and repeat the GET request. You should be able to see now the data in the dataset, like shown in Figure 4.35.

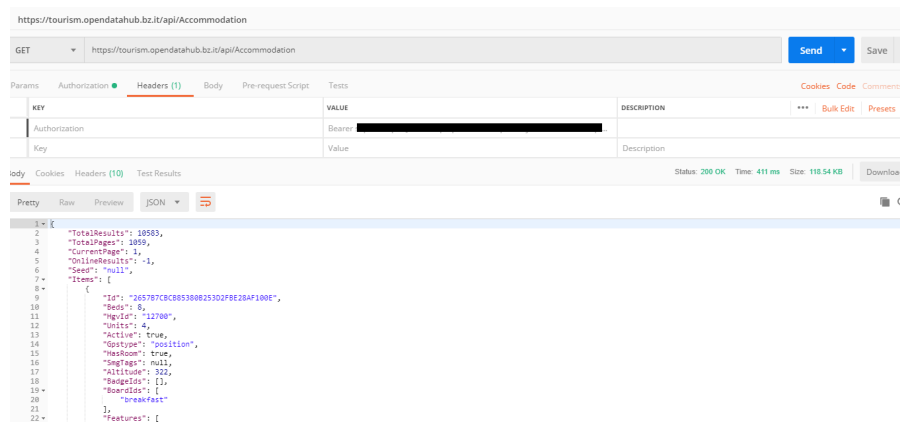


Figure 4.35: Access to data requiring authorisation.

Data Exporting

By default, queries to the Open Data Hub return data in JSON format and postman does not need any setup for that. It is however possible, for some datasets in the Tourism domain—check [Exporting and saving data](#) section for the list, to have postman receive data in CSV. The required set up is shown in [Figure 4.36](#): in the *Header* tab under the query, add a key **Accept** with value **text/csv**.

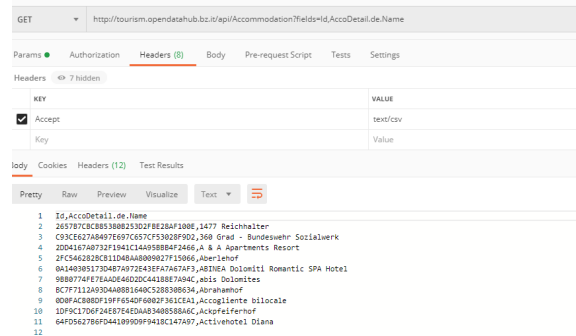


Figure 4.36: Exporting data from the Tourism domain in CSV format.

4.3.4 How to insert and modify NOI Events?

After reading this article, you will be able to use the Open Data Hub tourism portal to insert, modify, and delete events that take place at NOI Techpark in Bolzano (in the remainder, **NOI events**).

Preliminaries

Since you must login to create events, you need valid credentials to be able to add NOI events, that you should have received from the Open Data Hub team.

Go to <https://databrowser.opendatahub.bz.it> and click on Log in (top right corner)

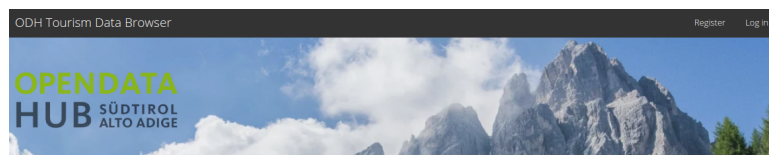


Figure 4.37: Upper section of the tourism portal.

Provide your credentials, then you will be redirected to your homepage, that shows among other information, the roles you have within the Open Data Hub.

Creation of a new NOI Event

Once logged in, click on *ODH Data* → *Events NOI* → *Events EURAC NOI* (see [Figure 4.38](#)).

Note: The drop-down menu that you will see might differ from those shown in the screenshot, depending on your permissions.



Figure 4.38: Menu item to create a new event

You will now see a list of events that will take place at Bolzano's NOI Techpark today or in the next days. For each event, the description, start and end date, and the location where it takes place are shown. If the event is marked as **Active**, it is displayed on the official NOI web page at <https://today.noi.bz.it/>.

 A screenshot of a web application showing a table titled 'Events EURAC NOI List'. The table has columns for 'Event', 'Start', 'End', 'Location', 'Status', and 'Action'. There are several rows of event data listed.

Figure 4.39: List of events.

In order to add a new event, click on the New button to create a new event.

In the dialog that opens, fill in all the fields you deem necessary, but at least the title, organiser, location, and time.

 A screenshot of a 'New Event' dialog box. It contains various input fields for event details: Title, Description, Location, Start Date, End Date, and a checkbox for 'Active'. There are also buttons for 'Cancel' and 'Save'.

Figure 4.40: An example event with a few details provided.

Remember to tick the *Active* and *noi.bz.it Active* checkboxes: The latter allows the event to show up on <https://today.noi.bz.it/>.

If the event is set to take place in more rooms, click on the Room Management button to add more rooms and time slots to the event.

If the event has a web page and/or a video trailer, you can add a link to them in the *Web Page (URL)* and *Video (URL)* text-fields.

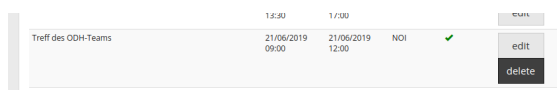
It is even possible to add images to the event, by clicking on the *Images* tab on top of the dialog and then on Choose File to upload a file. For each image, a few information can be added:

- The author's name.
- The licence used for the image, either **Proprietary** or **CC0**. 

Hint: We prefer that a **CC0** licence be used; it is necessary to have the rights to upload the photo with **CC0**.

- The position of the image within the gallery, if you upload more than one image. Image in position **0** will be the cover page of the gallery

When you have provided all the necessary information, click on Create to create the event, which will now show up in the list.

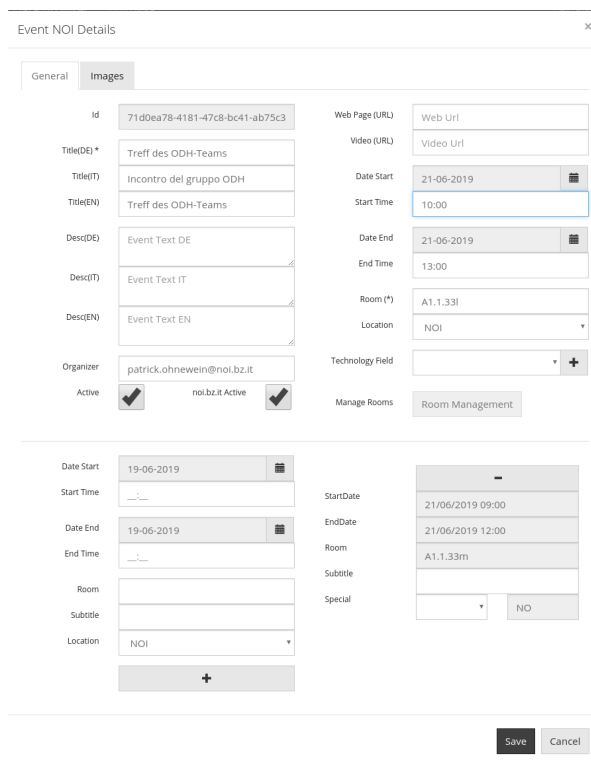


	12:00	1/300			
Treff des ODH-Teams	21/06/2019 09:00	21/06/2019 12:00	NOI	✓	edit delete

Figure 4.41: List with the new event.

Note that the title of the event is shown in the list in the language selected in the GUI (German in [Figure 4.41](#)).

If you later need to modify the event, click on the Edit button next to the event in the event list. For example, suppose the event used throughout this howto needs to be modified, because the meeting had to be postponed by one hour (10:00 to 13:00, instead of 9:00 to 12:00). Also the room is not available anymore, therefore it must be changed as well. These changes are shown in picture [Figure 4.42](#).



Event NOI Details

General Images

Id: 71d0ea78-4181-47c8-bc41-ab75c3

Title(DE) *: Treff des ODH-Teams

Title(IT): Incontro del gruppo ODH

Title(EN): Treff des ODH-Teams

Desc(DE): Event Text DE

Desc(IT): Event Text IT

Desc(EN): Event Text EN

Organizer: patrick.ohnwein@noi.bz.it

Active: ☒ noi.bz.it Active: ☒

Web Page (URL): Web Url

Video (URL): Video Url

Date Start: 21-06-2019

Start Time: 10:00

Date End: 21-06-2019

End Time: 13:00

Room (*): A1.1.33I

Location: NOI

Technology Field: +

Manage Rooms: Room Management

Date Start: 19-06-2019

Start Time: --:--

Date End: 19-06-2019

End Time: --:--

Room:

Subtitle:

Location: NOI

StartDate: 21/06/2019 09:00

EndDate: 21/06/2019 12:00

Room: A1.1.33m

Subtitle:

Special: NO

Save Cancel

Figure 4.42: Changing event's details.

Click on Save to save the modified event.

To delete an event, click on the Delete button next to the event, then confirm your choice in the confirmation dialog that will appear.

4.3.5 How to Publish your Web Component on the Open Data Hub Store

This how to guides you in the process of making available a Web Component that you created with the Open Data Hub Community, by publishing it on Open Data Hub Store.

The only requirement is that the Web Component **must** be released as an *Open Source Licence*.

One of three alternatives can be chosen to publish a Web Component on the Store: as a **full open source** project, as a **forked** project, or using what we call the **external workflow**; the preferred being the first one.

1. When using the **full open source** path, you simply hand in the source code of your Web Component to the Open Data Hub team, no additional effort is required from your side. The code will be placed in a GIT repository, and immediately made available through the Store. Future versions of the Web component are developed under the control of the Open Data Hub team directly within this repository.
2. The **forked** project way will still see the Web Component's source code saved in a GIT repository, but you own full control of it, and decide about its future versions. The difference with the previous method is that the updates are done by you in your repository and the Open Data Hub team will need to keep its copy synchronised with yours.
3. Finally, the **the external workflow** is the one in which nothing is saved in a GIT repository. You will maintain full control of the source code and of the Web Component's development, including the right to pull it out of the store. Should you decide to follow this path, you will have to satisfy a few more requirements, to ensure proper integration with the Store:
 1. You need to install in the root directory of your Web Component's source code a suitable `wcs-manifest.json` file, that you will receive via pull request.
 2. Your Web component must be saved (better if in *minified* form) under a `dist` directory, e.g., as `$ROOT/dist/widget.min.js`, where `$ROOT` is the root directory of your repository.
 3. You need to tag a commit on your master branch with a *semantic versioning* tag, to communicate to the Store that the corresponding version should be published (example, `git tag -a "v1.2.3" -m "v1.2.3"`).

An example of these files and setup can be found in the published example of a generic map, that you can find at <https://github.com/noi-techpark/webcomp-generic-map>.

Finally, to ensure that your Web Component is kept updated in the Store, suitable pull requests will be sent to your repository.

In all three cases, a repository called **origins** (hosted on [github](#)) will hold a reference to each Web Component's repository, from where all files that are necessary for its deployment including the *manifest* file, the javascript, and logo if present to each Web Component's version. It is therefore important, in case the **external workflow** has been chosen, that the URL to the Web Component be stable, otherwise the Web Component will *not* be available.

4.3.6 How to Access Open Data Hub Data Using SPARQL

The Open Data Hub's dataset can be queried using the SPARQL query language, using the [Open Data Hub Knowledge Graph Portal](#). This howto helps you in getting acquainted with the functionalities offered by the endpoint. However, this howto does not cover SPARQL: if you are not familiar with it, here is some reference:

- The [SPARQL Query Language Recommendation](#) is the official and normative W3C definition of SPARQL and also contains a lot of examples and queries to learn from
- A [tutorial about SPARQL](#) written by Apache Jena's team. Oriented toward Jena, it nonetheless includes and explains a lot of basic notions

Data Available in the Portal

The landing page of Open Data Hub's SPARQL endpoint contains the following elements:

1. The buttons in the banner at the top of the page.

Playground

The *Playground* is a space in which to freely write SPARQL queries against the Open Data Hub datasets. It is most suited for users that already know SPARQL and how to use it to interact with Open Data Hub.

See section *Working in The Playground*.

Regular Queries

Regular Queries are a sample queries that can be used either standalone, to gather example data, or can be edited and modified to tweak the results.

See section *Working with Regular Queries*.

Data Quality Queries

Similar to Regular Queries, *Data Quality Queries* are precooked queries that will gather data, but with an emphasis on their quality. They can be used to check whether some of the data are incomplete.

See section *Working with Data Quality Queries*.

Mobility

Mobility queries are sample queries against all the datasets in the entire mobility domain. They can be used as they are or modified and tweaked to extract more precise data.

You can refer to section *Working in The Playground*.

Tourism and Mobility

Tourism and Mobility queries combine datasets from the tourism domain with observations gathered by sensors in the mobility domain.

You can refer to section *Working in The Playground*.

2. The main area, consisting of a large textarea, in which to write SPARQL queries, and of a number of precooked queries when the *Regular Queries* or *Data Quality Queries* buttons are clicked. The three buttons on the textarea's top right corner can be used to
 - Copy the URL of the query and share it, store it for future use, or use it in scripts.
 - maximise the textarea
 - execute the query. If the query contains some syntactic error, it is accompanied by a yellow question mark and it is not executed, but an error message is displayed
3. A number of visualisation and download options in the bottom area. Also this part of the area can be maximised
 - *Table*. A simple table with a result on each row
 - *Response*. The actual JSON received as result
 - *Pivot Table*. Analyse statistically the query result
 - *Google chart*. Use the data retrieved within a Google Chart. The default representation is a simple table, more can be employed, by clicking on the Chart Config button on the right-hand side.
 - *Geo*. See on a map the location of the results
 - download the result set as a CSV file

Working in The Playground

The playground is the place in which you can build you queries against the Open Data Hub endpoint. Queries can be built using built-in or custom prefixes as well as all SPARQL operators. There is a validation of the queries, therefore in case of mistakes a red warning icon will appear on the left-hand side of the offending line.

Note: Generic queries might return hundreds or thousands of results, so the use of the LIMIT clause helps to receive quicker answers.

Working with Regular Queries

Regular queries are predefined queries that give a glimpse of the data contained in the Open Data Hub. Regular queries are rather generic and can be used as starting point for more precise and refined queries. They can be edited directly in the textarea or copy and pasted in the Playground.

Working with Data Quality Queries

Data quality queries are built with purpose to verify if there are incomplete or wrong data in a dataset.

4.3.7 How to Access Open Data Hub Data With R and SPARQL

Datasets and their data in the Open Data Hub can be accessed using R, a software for statistical analysis.

This howto shows you a method to retrieve data from the Open Data Hub, but does not address other features like, for example, plotting fetched data on a map.

It is also assumed you have already installed R on your workstation as well as the required R's [SPARQL library](#) from a CRAN (Comprehensive R Archive Network) mirror.

Install SPARQL library

To install the SPARQL library, simply execute on Debian-like systems:

```
~# Rscript -e "install.packages('SPARQL')"  
Installing package into '/usr/local/lib/R/site-library'  
(as 'lib' is unspecified)  
trying URL 'https://cloud.r-project.org/src/contrib/SPARQL_1.16.tar.gz'  
Content type 'application/x-gzip' length 6548 bytes  
=====  
downloaded 6548 bytes  
  
* installing *source* package 'SPARQL' ...  
** package 'SPARQL' successfully unpacked and MD5 sums checked  
** R  
** byte-compile and prepare package for lazy loading  
** help  
*** installing help indices  
** building package indices  
** testing if installed package can be loaded  
* DONE (SPARQL)
```

(continues on next page)

(continued from previous page)

The downloaded `source` packages are `in`
`‘/tmp/RtmpISkL9Z/downloaded_packages’`

If you see the message `* DONE (SPARQL)`, the installation was successful.

If you see instead any **ERROR**, like those reported below, please refer to section *Troubleshooting*:

Warning messages:

```
1: In install.packages("SPARQL") :
  installation of package ‘RCurl’ had non-zero exit status
2: In install.packages("SPARQL") :
  installation of package ‘SPARQL’ had non-zero exit status
```

Documentation for the library can be found in the library’s [PDF documentation](#)

In order to fetch data, you need:

1. An endpoint, which for Open Data Hub is <https://sparql.opendatahub.bz.it/sparql>
2. a SPARQL query, that you can simply copy from one of the precooked queries at <https://sparql.opendatahub.bz.it/>
 We’ll be using this one:

```
PREFIX schema: <https://schema.org/>
PREFIX geo: <http://schemas.opengis.net/geosparql/1.0/geosparql_vocab_all.rdf#>
PREFIX noi: <https://noi.example.org/ontology/odh#>

SELECT ?pos ?posLabel
WHERE {
  ?p a noi:Pizzeria ;
    geo:asWKT ?pos ;
    schema:name ?posLabel ;
    schema:geo ?geo .
  FILTER (lang(?posLabel) = "it")
}
LIMIT 10
```

3. An R script to put all together

```
1 library(SPARQL)
2
3 endpoint <- "https://sparql.opendatahub.bz.it/sparql"
4
5 query <-
6 'PREFIX schema: <https://schema.org/>
7 PREFIX geo: <http://schemas.opengis.net/geosparql/1.0/geosparql_vocab_all.rdf#>
8 PREFIX noi: <https://noi.example.org/ontology/odh#>
9
10 SELECT ?pos ?posLabel
11 WHERE {
12   ?p a noi:Pizzeria ;
13     geo:asWKT ?pos ;
14     schema:name ?posLabel ;
15     schema:geo ?geo .
16   FILTER (lang(?posLabel) = "it")
```

(continues on next page)

(continued from previous page)

```

17 }
18 LIMIT 10'
19
20 result_set <- SPARQL(endpoint,query)
21 print(result_set)

```

The script above can be saved in a file called `R-demo.r` and executed using the **Rscript R-demo.r** command. The output will be similar to:

```

~# Rscript R-demo.r
Loading required package: XML
Loading required package: RCurl
$results
                                     pos
1  "POINT (11.440394 46.511651)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
2  "POINT (11.200728 46.729921)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
3    "POINT (11.9412 46.9803)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
4    "POINT (11.4278 46.4135)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
5  "POINT (11.326362 46.310963)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
6  "POINT (12.279453 46.733497)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
7  "POINT (10.867335 46.622179)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
8  "POINT (11.241217 46.246141)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
9  "POINT (11.598339 46.40688)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
10  "POINT (12.0114 46.7474)"^^<http://schemas.opengis.net/geosparql/1.0/geosparql_
↪vocab_all.rdf#wktLiteral>
                                     posLabel
1      "Ristorante Pizzeria Bar Pirpamer"@it
2      "Bar Pizzeria Alpenhof"@it
3      "Ahrner Wirt Ristorante Pizzeria"@it
4      "Ristorante Pizzeria Adler"@it
5      "Hotel Al Mulino"@it
6      "Ristorante Pizzeria Zentral"@it
7      "Hotel Ristorante Bar Rasthof Vermoi"@it
8      "Hotel Grünwald"@it
9      "Hennenstall"@it
10 "Après Ski Bar Pizzeria Ristorante "Gassl""@it

```

In the script, all data fetched are kept into the **result_set** variable and can be manipulated at will using R libraries.

Troubleshooting

SPARQL installation fails!

When installing a package, R tries to satisfy all the package's dependencies and installs any missing library required by the package. If you still stumble upon errors, like for example:

```
Warning messages:
1: In install.packages("SPARQL") :
  installation of package 'RCurl' had non-zero exit status
2: In install.packages("SPARQL") :
  installation of package 'SPARQL' had non-zero exit status
```

It means that SPARQL's dependency **RCurl** also failed. In this case it is not easy to spot the root cause, which is a missing package in the OS installation, called **libcurl4-gnutls-dev**. To install it on a Debian-like system, use as *root* the following command:

```
~# apt-get install libcurl4-gnutls-dev
```

I have some strange warning when executing the script!

If you execute a query and the outcome is not a result set but some error message similar to the following ones, please verify that the URL of the SPARQL endpoint is correct: **<https://sparql.opendatahub.bz.it/sparql>**

```
Opening and ending tag mismatch: meta line 5 and head
Opening and ending tag mismatch: meta line 4 and html
Premature end of data in tag meta line 3
Premature end of data in tag head line 2
Premature end of data in tag html line 1
```

RESOURCES FOR DEVELOPERS

This section contains all information that is necessary to developers that want to collaborate with the Open Data Hub team (e.g., developers that send pull requests to the Open Data Hub repositories) or are contracted to write code or apps for the Open Data Hub project (Open Data Hub Core Hacker)

5.1 Guidelines for Developers

Open Data Hub is a collection of software, databases, and services coordinated and hosted by [NOI techpark](#) in Bolzano/Bozen, South Tyrol. Currently, Open Data Hub systems are related to mobility and tourism. In the future Open Data Hub might diversify into more fields.

Companies and developers contributing to Open Data Hub must follow the guidelines listed in the documents as close as possible.

The aim of the Open Data Hub Developer's Guidelines ("**The Guidelines**") is to simplify the hosting and maintenance of the software, databases, and services by the Open Data Hub developers and maintainers at NOI Techpark ("the Open Data Hub team").

The Guidelines describe the conventions to which a developer must adhere, to be able to become an active Open Data Hub developer or to see his work being incorporated into the Open Data Hub. They are split in two parts:

- **Platform Guidelines** explain the preferred programming languages, how to expose the data after you manipulated them, the use of third-party libraries or plugins, and so on.
- **Database Guidelines** clarify how to design a database that shall become part of the Open Data Hub platform.

Both of them are summarised in the remainder of this section, and can be found in full version in the pages [Platform Guidelines - Full Version](#) and [Database Guidelines - Full Version](#) respectively

5.1.1 Platform Guidelines - Bignami Version

The *Platform Guidelines* contain the software and programming language requirements, coding conventions, and directions for development. This section contains **only** the most important points.

Please check the full version of this document at [Platform Guidelines - Full Version](#) if you want to know more details, if you have some doubt or if what you were looking for is not mentioned in this summary.

- Programming Language is **Java**, in its latest or second to last version.
- The source code **must be documented** according to the [Javadoc style guide and tags](#).
- Java components of Open Data Hub can be developed as **libraries**, **standalone applications**, or **server applications** running in Apache Tomcat.

- The source code is **built nightly**; **build configuration** should be provided in either Ant or Maven (preferred), Makefile, or shell script.
- **Third party libraries** can be used, provided they are established, FOSS-licenced, and do not overlap functionalities. This applies also to third party libraries used in application developed in other languages.
- Front-end applications can be deployed in **Javascript**, version EC 2015, and must support modern browsers.
- **Node.js** can be used to deploy headless or server applications.
- Web front ends use the **latest HTML and CSS** versions, must work on mobile devices (responsive design) and should implement some basic accessibility principle.
- **JSON** must be used as exchange language, while **XML** is welcomed as well.
- The latest or second to last version **Apache Tomcat** is used to run server application; only **API/REST end points** have direct access to the database server.
- There's **no file system persistence**, everything must be stored in the DB. JDBC data source and passwords should be stored in environmental variables.
- Pay attention to **RAM usage**, applications will undergo **load testing**.
- **PostgreSQL** RDBMS (Relational DataBase Management System) is used, but not in its recent release (expect to use 2-3 versions before the latest), **PostGIS** spatial extension is required as well.
- Developers will have an unprivileged role to access the DB and must follow best practices to query the DB from Java/Javascript.

5.1.2 Database Guidelines - Bignami Version

The *Database Guidelines* contain the database design and database programming principles along with software version requirements. This section contains **only** the most important points.

Please check the full version of this document at [Database Guidelines - Full Version](#) if you want to know more details, if you have some doubt or if what you were looking for is not mentioned in this summary.

- The database can be designed with one of the **Relational Model**, **Object-Relational Mapping (ORM)**, or **Semi-structured Data** methodologies.
- A database designed with either methodology must be shipped with DDL (Data Definition Language) - *schema files* containing the **CREATE** statements.
- Each database must include a **version table** and **indices** on tables.
- All (SQL) source code must be well-documented, with in-line comments and higher level documentation.
- Use standard database features - Sequences, primary and foreign keys, constraints (unique, check, not null), default values, views, and so on and so forth.
- Separate business logic from database design; avoid stored procedure as much as possible.
- Small procedures and functions, if needed. must be written in **PL/pgsql**.
- Do **not** use foreign data wrappers.
- Consider using declarative partitioning for large tables - and contact Open Data Hub team beforehand to discuss it.
- Always use UTF8 character encoding and do **not** override it.
- Default collation is **en_US**, which works well for German and Italian as well.
- Never use money type, but numeric.

- Dates and time stamps must be store to avoid ambiguity. Never store them as text, but rather use their data types, `date` (in UTC format) and `timestamp with timezone`. Unix timestamp is accepted as well.
- When using or manipulating JSON data always follow **ISO_8601** standard.

5.2 Platform Guidelines - Full Version

Changelog

- **2020-01-20 version 1.1** – fixed outgoing links
- 2018-05-28 version 1.0
- 2018-03-30 version 1.0-beta

This document represents **Part 1** of the guidelines and presents the preferred programming languages, databases, and protocols to be used, data exchange and exposition methods, coding conventions, and regulates the use of third-party libraries.

There are scenarios where an exemption from the guidelines is acceptable. The following is a **non-exhaustive** list of such scenarios.

1. **Use of foreign technologies.** The development of a Open Data Hub component requires the use of platforms, languages or generally technologies that are different from the ones listed in the guidelines. An example might be a component that depends on an already developed custom library written in a programming language not listed in the guidelines.
2. **Use of technologies that are not mentioned in the guidelines.** Future Open Data Hub component might require technology that is not listed at all in the guidelines. An example is a component that must be hosted on specific hardware needed for machine learning platforms.

A Open Data Hub contributor who runs into such a scenario must contact the Open Data Hub team to discuss that specific scenario. If the exemption is reasonable and can be motivated the Open Data Hub team will agree and allow it. To avoid misunderstandings, contributors must expect to get a **written statement** about such a decision.

Note: If you can not find any answer to your question or doubt in this document, please contact the Open Data Hub team or open an issue in the [github repository](#) of this document.

5.2.1 Programing Languages, Environments, and Related Technologies

Java

The chosen programing language for Open Data Hub is Java, more precisely the Java Platform, Standard Edition (Java SE).

Source code will be compiled with either the [OpenJDK](#) or the [OracleJDK](#), which share the same code base anyway. Resulting binaries will run in the corresponding JVM.

Java Version

Java is generally backwards-compatible, so code written for a previous version of the JDK will likely compile on the next compiler version and run on the next JVM version. However, contributors ought to use a reasonably modern version of Java in order to avoid deprecation warnings and make use of modern language features.

Contributors can expect the Open Data Hub team to use the **current stable version** of the language. Of course, a certain delay is to be expected between the time a Java release becomes generally available and the time OS vendors and hosting providers make it available. This delay, that can easily be in the order of one year, must be taken into account.

Environments

Open Data Hub Java components can be developed as:

- Java **libraries**.
- Java **standalone applications**, running headless.
- Java **server applications** running in Apache Tomcat:
 - API/REST end points.
 - Web applications.

More information about standalone and server applications can be found under section *Platforms and Architectural Considerations*.

Open Data Hub components **must not** be developed as fat clients (like e.g., Swing, SWT). **Web applications** are the preferred technology.

While native Android applications can be developed in Java, they should also be avoided as they are not a cross platform solution (Android vs. IOS). For the mobile space, (mobile) web applications or cross platform environments based on JavaScript are preferred (see section *JavaScript*).

Documentation

Source code must be commented following the established *Javadoc style guide and tags*.

Complex section of the code (for example not-trivial algorithms) must have dedicated comment sections.

Higher-level documentation must be available as well and if possible, it **must be kept** in a simple, text-based format, such as plain text, Markdown or HTML. The rationale behind this choice is that these formats - unlike binary file formats such as ODT or DOCX - can be versioned in a source code management system.

Builds

The Open Data Hub team runs automatized nightly builds (and tests) of Open Data Hub software components. It must therefore be possible to rebuild the binaries (JARs or WARs) starting from the source code all the way down to the complete binaries in a headless environment.

Developers **must provide** standard build configurations for one of the usual Free / Open Source Software (“FOSS”) build tools used in the Java space (such as Maven or Ant). Alternatively a simple Makefile or shell script (the nightly build system runs on Linux) will suffice.

Considerations about testing are described in another document.

Use of Third-Party Libraries

Most Java projects use one or more third-party libraries. Regarding the use of such libraries in Open Data Hub, the following guidelines apply:

- The library must be stable, well known and well supported.
- The library must be distributed under a FOSS license.
- Avoid creating pile-ups of libraries with overlapping functionality.

JavaScript

While the primary programming language for Open Data Hub is Java, there are use cases where JavaScript is accepted or even dictated by the environment (like e.g. web front ends).

The Open Data Hub team endorses the language revision **ECMAScript 2015** (a.k.a. ES 6) and encourages a modern, expressive use of the language (e.g. block scoped variables, function expressions, promises and many more).

The usage of JavaScript falls into the two categories: Web front ends and Node.js, as detailed in the next sections.

JavaScript Web Front Ends

Most modern web applications will use JavaScript in the web front end. The Open Data Hub team is agnostic about how the front end is implemented (classic web application vs. single page web application).

In the likely case that JavaScript front end libraries and frameworks are used, the following guidelines apply:

- The library or framework must be stable, widely used and well supported - avoid using cutting edge libraries with APIs that are not settled yet.
- The library or framework must be distributed under a FOSS license.
- The library or framework must be cleanly imported into the project with one of these methods:
 - By means of a JavaScript package manager with a configuration file (such as **npm** and **package.json**).
 - Manually, by using a clearly labelled *include path* (such as `import /vendor/name/version/file.js`).

To avoid having to support many programming languages, source code **must not** be developed in a transpiled language (e.g. TypeScript or CoffeeScript),

In terms of browser compatibility, developers can use ES 2015, as said. According to the [ECMA Compatibility table](#), ES2015 is well supported in all modern browsers (Chrome, Firefox, Safari, Edge) both in desktop and mobile version.

Generally speaking, support of legacy browsers (MS Internet Explorer) is not an issue. Cross-browser testing is, of course, still necessary and expected.

If a build system such as [webpack](#) is needed, its use must be clearly documented as the Open Data Hub team must integrate it into their nightly builds system.

JavaScript Running in Node.js

Besides the front end, JavaScript code can be also used for headless or server applications, provided they have limited complexity.

In case the developer needs to create large pieces of business logic or complex web applications, Java ought to be the preferred environment.

Most front end guidelines mentioned in the previous section apply here as well, in particular those about *libraries*. A complete `package.json` file is a must here. It is required that the Node.js project be installed simply by running **npm install**.

Use cases for Node.js in the Open Data Hub are:

- Simple REST end points.
- Simple web applications.
- Tools that operate on JSON data.
- Scripting / glue code.

The Open Data Hub team generally uses an [LTS release](#) of Node.js, adopted soon after it becomes available, although some time might be needed for the hosting provider to make it available.

SQL

See section [PostgreSQL](#) below.

HTML and CSS

Web front ends are, of course, developed using HTML and CSS in their current versions.

It is important that all web pages render correctly in all modern browsers (Chrome, Firefox, Safari, Edge).

Generally speaking, support of legacy browsers (MS Internet Explorer) is not an issue. Cross-browser testing is, of course, still necessary and expected. A minimum requirement is that all HTML validates against [the W3C validator](#).

As most web traffic is nowadays coming from mobile devices, all general purpose web UIs exposed to end users should be implemented to work well on mobile devices by using standard techniques, such as **responsive design**.

In the development of the web front-end, Accessibility principles should be taken into account when designing web pages.

XML and JSON

XML and **JSON** are both important data description languages, heavily used in the context of Java, JavaScript, web applications, and APIs; therefore they are both used and welcome in the Open Data Hub.

JSON is of particular interest as that is the preferred data exchange format for REST endpoints. It also plays a role in the persistence layer, as Open Data Hub allows the use of JSON records in PostgreSQL tables (see section [PostgreSQL](#) below).

5.2.2 Platforms and Architectural Considerations

Java server applications running in Apache Tomcat

[Apache Tomcat](#) is a well established, light weight FOSS web server that implements among others the Java Servlet specification.

The Open Data Hub team generally uses the latest or second to last release of Tomcat, to run Java server applications in the previously mentioned contexts:

- API/REST end points.
- Web applications.

The desired design is that **only API/REST end points** directly access the database server, while web applications just talk to the API/REST end points.

Automatic Deployment

Each Tomcat instance normally runs a few web applications, hence expect a Open Data Hub web application's WAR file to be bundled together with other WAR files to run on a given instance.

The automatic build systems takes care of this bundling and deploying. It is therefore very important that all WARs can be build automatically, as mentioned in the [section about Java](#).

No File System Persistence

Currently, the Open Data Hub team uses Amazon Web Services for Tomcat hosting, in particular the managed service known as *Elastic Beanstalk*. While there is no hard dependency on this provider -that could be changed at any point in the future, the architectural design of Elastic Beanstalk has partly modelled/shaped the engineering choices of the Open Data Hub team in the design of its web application.

First and foremost, servers are considered volatile. This means a Open Data Hub component running in Tomcat **can not expect** to see a persistent file system!

All web applications must therefore be developed with the database as **the only persistent storage layer**. This architectural choice has a few advantages:

- Web applications can be distributed over more than one web server (horizontal scaling), increasing availability and performance.
- Backup and disaster recovery is very much simplified - a failing instance can just be replaced by a new instance and the application can be deployed again.

Developers must pay particular attention to this point: **There is no persistent file system**. Hence no changeable configuration files, no application specific log files. Everything is stored in the database.

Data Source

One subtle point is the question “*Where is the JDBC data source and password stored?*”. It cannot be stored in a file and it must not be stored in the source code or context files. The recommended way to store this information is in Java environment properties.

The system will set these variables when launching Tomcat:

```
JDBC_CONNECTION_DRIVER=org.postgresql.Driver
JDBC_CONNECTION_STRING=jdbc:postgresql://host:5432/db?user=username&password=secret
```

The developer can then read them with:

```
System.getProperty("JDBC_CONNECTION_DRIVER");
System.getProperty("JDBC_CONNECTION_STRING");
```

RAM Usage

The Open Data Hub encompasses a considerable number of web applications that are bundled together to run on a few Tomcat server instances. Contrary to popular belief, RAM is not an infinite resource. Contributors are kindly reminded to pay attention to the RAM usage of their web applications, since load testing is expected.

Java standalone applications, running headless

Besides wapplications running in Tomcat, the Open Data Hub also has headless standalone applications written in Java or JavaScript/Node.js.

These are meant for special use cases, such as compute intensive jobs or batch processing, made upon request.

Almost everything said in the previous section about Tomcat, applies here as well.

Again, the preferred way to run these applications is in an environment where servers are volatile and the only persistence layer is the database.

PostgreSQL

PostgreSQL is one of the most established RDBMS on the market and is generally described as being by far the most advanced FOSS RDBMS and therefore it has been chosen as the primary database system for Open Data Hub.

There is a **new major release** of PostgreSQL per year and each release is supported for 5 years, according to [the versioning policy](#). Contrary to the case of the other products mentioned in these guidelines, the Open Data Hub team generally will **not run the latest** or even previous version of PostgreSQL. Expect the version available for Open Data Hub to lag about 2-3 years behind the latest available release.

Extensions

Most, if not all of the [extensions distributed with PostgreSQL](#), can be expected to be available, together with the third-party [spatial query extension PostGIS](#) is also available.

Other extensions are very likely **not available**, so ask the Open Data Hub team if in doubt.

Accessing the Database

Application developers will get one or more unprivileged database roles to access the database. Access will be done via JDBC when using Java, or via any of the available PostgreSQL modules for Node.js when using JavaScript.

The data source strings must be parsed from the environment variables (see section [Java server applications running in Apache Tomcat](#)).

The maximum number of concurrent database sessions will be generally limited per role, therefore each developer must clarify with the Open Data Hub team what an acceptable number is, depending on the application.

Since PostgreSQL will refuse a connection if that number is exceeded, developers must take this number into account, whether they configure a connection pool or not.

Open Data Hub databases generally are configured to accept connections only from the known hosts where the application servers are deployed.

Contributors must follow well known best practices when querying the database from Java or JavaScript:

- When processing large datasets, consider setting smaller values of `fetchsize` or equivalent parameter to avoid buffering huge result sets in memory and running out of RAM.
- When performing a huge number of DML statements consider switching off any client side autocommit feature and rather bundle statements into transactions.
- Do **not** open transactions without closing them, in other words, do **not** leave sessions in transaction!

Database Design and Usage

This section has been moved into its own document, [Database Guidelines - Full Version](#).

5.3 Database Guidelines - Full Version

Changelog

- **2020-01-20 version 1.1** – fixed outgoing links
- 2018-05-28 version 1.0

This document represents Part 2 of the Open Data Hub Developer's Guidelines and clarifies the database design criteria for developers who contribute their own databases designs to the Open Data Hub platform.

Basic information about the PostgreSQL versions, PostgreSQL extensions and how to access PostgreSQL from Java or JavaScript, intended for developers that contribute code that just uses an existing database, are explained in the [Platform Guidelines - Full Version](#) document as well. Please refer to that document for a general introduction to the scope of the present guidelines.

5.3.1 Database design methodology

The Open Data Hub team is generally agnostic about database design and acknowledges the existence of different design and development methodologies.

Specifically, the following methodologies are well known and acceptable:

1. **Relational Model.** The data schema is implemented using normalized relations with standard SQL concepts (schemas, tables, columns and keys). The **CREATE** statements are written by the developer.
2. **Object-Relational Mapping (ORM).** The underlying data schema is based on the relation modal, but the **CREATE** statements are generate by an ORM framework that automatically maps entities to relations.
3. **Semi-structured Data.** Entities are stored in a semi-structured format. For the Open Data Hub the preferred format is JSON. Specifically, the recommended design is to map each entity to its own table. The table should have at least two columns: one traditional ID column and one JSON data column. The (simple) **CREATE** statements are written by the developer. The JSON data column must use the PostgreSQL native data type **jsonb** (see [binary stored JSON](#) in PostgreSQL documentation).

PostgreSQL supports all three methodologies well. It is also possible to have a hybrid design mixing 1. and 3.

A developer contributing a database design to Open Data Hub must provide the DDL , a.k.a. *schema files* containing the **CREATE** statements.

Like all source code files, the *schema files* must be commented in-line and accompanied by additional, higher level documentation.

Besides source code file comments, database objects must also be commented with the SQL **comment** command (see [Sample Code 1](#) below).

Updates must be provided in the form of **ALTER** statements, so the modifications can be easily applied to existing databases (see [Sample Code 2](#) below).

All database designs should contain a version table, where the version is stored (and updated with each update).

The Open Data Hub team likes to stress this point: **do not just commit database schema dumps**, but rather treat SQL-DDL files as source code and cleanly distinguish the initial creation and later updates.

Sample Code 1: A DDL source file called `foo.sql`

```
-- foo.sql
-- a document with appendices
--
-- changelog:
-- version 1.0
--
-- copyright, author etc.

create sequence foo_seq;

create table doc (
    id          int default nextval('foo_seq'),
title    text not null,
body     text,
primary key(id)
);

comment on table doc is 'stores foo documents';
```

(continues on next page)

(continued from previous page)

```

create table appendix (
    id          int default nextval('foo_seq'),
section char(1) not null,
body      text,
doc_id int not null,
primary key(id),
foreign key (doc_id) references doc(id)
);

comment on table appendix is 'stores appendices to foo documents';

create table foo_version (
    version varchar not null
);

insert into foo_version values ('1.0');
```

Sample Code 2: Update to schema of *foo.sql*, version 2.0:

```

-- foo.sql
-- a document with appendices
--
-- changelog:
-- version 2.0 - added a field
-- version 1.0
--
-- copyright, author etc.

BEGIN;

alter table doc add column publication_date date default current_date;

update foo_version set version = '2.0';

COMMIT;
```

The explicit transaction (**BEGIN - COMMIT**) will make sure the DDL update is applied cleanly or not at all. Note that DDL statements in PostgreSQL are transactional.

If methodology 2 (ORM) is chosen, the contributor should provide the cleanest DDL output the framework provides.

Contributors can expect their database design to be stored into a schema whose name is determined by the Open Data Hub team and executed as a non-privileged user account that has the given schema in its default **search_path** (see [DDL schema path](#) in PostgreSQL documentation).

Unless there is a specific reason, contributed designs must use **only a single schema** without using its explicit name, because that will be determined by the **search_path**.

Contributors are invited to make good use of standard database features, including -but not limited to:

- Sequences.
- Primary and foreign keys.
- Unique constraints.
- Check constraints.

- Not null constraints.
- Default values.
- Views.

5.3.2 Stored procedures and functions, foreign data wrappers

The Open Data Hub team would like to avoid stored procedures and functions as far as possible. **Business logic** should be implemented in the middle tier, **not** in the database system.

Hence, the general rule is that database designs submitted to the Open Data Hub **must** not contain business logic operations.

However, (small) utility procedures and functions, especially with respect to triggers, are allowed. When used, these procedures and functions must be written in [PL/PgSQL](#). Other server-side languages, even the trusted ones, are neither allowed, nor can they be expected to be available.

An example of such an allowed instance of a procedure is an audit trigger that, for any changes made to **Table A** generates a log entry that is stored in **Table B**.

Foreign data wrappers ([SQL/MED](#)) **must not** be used.

5.3.3 Indices and Partitioning

The submitted database designs must include creation of indices on tables.

Of course, the Open Data Hub team will monitor database performance and might be able to add indices at a later time. However, not anticipating obvious index candidates is considered a bug.

The database design contributor knows best what tables and what columns will benefit from indices, when the number of records grows.

In particular, if methodology 3 (JSON) is chosen, PostgreSQL provides specialized multi-dimensional indices of type GIN to index the [jsonb data type](#).

If the contributor anticipates designs with large tables (say more than 100M records or more than 5 GB on disk) and expects queries needing to sequentially scan those tables, **declarative partitioning** should be considered. The contributor must then contact the Open Data Hub team to agree on a declarative partitioning scheme in advance.

5.3.4 Encoding, collation and localization

All Open Data Hub PostgreSQL databases use the UTF8 character encoding as default encoding and this **must not be overridden** by a database design contributor.

The Open Data Hub team wishes to avoid any character encoding issues by using UTF8 for everything.

The *default collation* is en_US. For PostgreSQL running on Linux this collation already behaves reasonably for German and Italian:

```
select * from t order by s collate "en_US";
t
---
A
À
Ä
```

(continues on next page)

(continued from previous page)

B
(4 rows)

A contributor is free to add a custom collation such as `de_DE` or `it_IT`, either at the DDL level or the query level (see [PostgreSQL documentation on collation](#)), although there is most likely no need to apply other collations.

A database design **must not** use the money type. Currency amounts must be stored in fields of type `numeric` and the currency must be stored separately.

One important aspect concerns **dates** and **timestamps**.

Since the Open Data Hub applications span multiple regions and time zones, it is very important to be precise about date and time formats and time zone information.

Dates must be stored in the appropriate `date` data type. Dates stored in this data type will be automatically converted into the client native format when queried. **Never store dates as text** because this creates ambiguity. For example, what date represent the string `10-07-2018`? Is it the seventh of October 2018 or the tenth of July 2018?

The same holds true for timestamps that must be stored in the appropriate `timestamp` data type. Besides avoiding format ambiguities, this data type also includes also the time zone.

Note: PostgreSQL supports also a `timestamp without time zone` data type, according to the SQL standard. However, this data type **must not be used** as it does not store the vital time zone information.

Here ist the output of two queries executed almost at the same time on two PostgreSQL servers running in different time zones.

This is UTC (no daylight saving).

```
# select now();
      now
-----
2018-05-28 00:28:25.963945+00
(1 row)
```

And this is CET (with daylight saving), 2 hours ahead of UTC:

```
# select now();
      now
-----
2018-05-28 02:28:27.121242+02
(1 row)
```

You can see that these two queries were executed (almost) at the same time thanks to the time zone information (**+00** vs. **+02**). Without time zone information, the two time stamps appear as separated by two hours.

Note: When using the `date` and `timestamp` data types there is no format issue at all, as the PostgreSQL client libraries automatically convert from and to the client native format. For example a Java `Date` object is automatically converted to an SQL date value.

Sometimes developers need to convert to and from text. In case a contributing developer wishes to do this using PostgreSQL functions, they must use functions `to_date()` and `to_char()` (see [PostgreSQL documentation on function formatting](#)).

For example:

```
-- insert into date field d converting from German text:
# insert into dates (d) values (to_date('28.5.2018', 'DD.MM.YYYY'));

-- select date field d and convert to German text:
# select to_char(d, 'DD.MM.YYYY') from dates;
to_char
-----
28.05.2018
(1 row)
```

Sometimes timestamps are stored as numbers, the so called Unix time stamp (see [unix timestamp](#) on wikipedia).

This is also acceptable, as the Unix time stamp always follows UTC and is therefore unambiguous.

For JSON data, contributors must make sure that the textual representation of dates and timestamps follow the ISO standard [ISO_8601](#) (see more [on Wikipedia](#)). Examples:

- “ts”:”2018-05-28T00:54:28.025Z”
- “d”:”2018-05-28”

PostgreSQL accepts these strings as inputs for `timestamp` and `date` types even as text (there is an implicit type cast).

Also note JavaScript has a `Date.prototype.toISOString()` method.

5.4 Development, Testing, and Production Environments

Note: Information in this section is still provisional!

Figure 5.1 shows the various environments which compose the whole Open Data Hub development process.

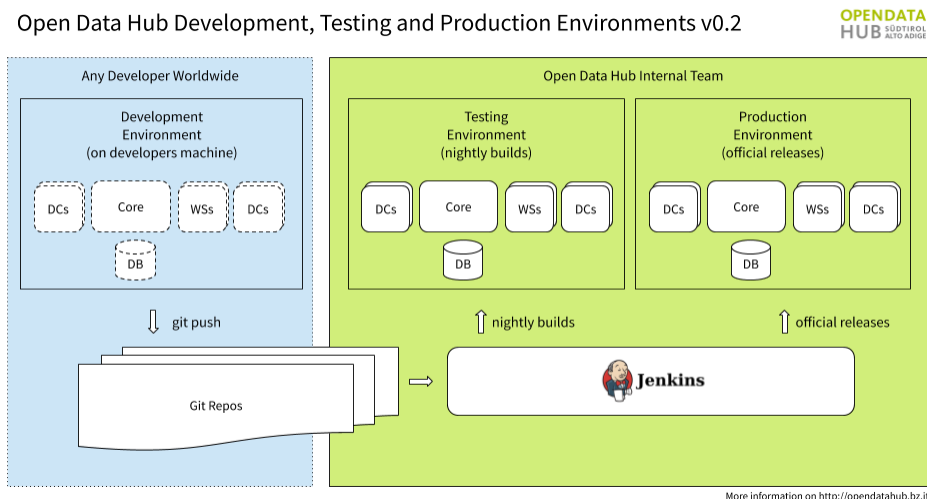


Figure 5.1: Diagram showing the development, testing, and production environments in the Open Data Hub project.

On the right-hand side, the internal structure of development is shown, while on the left-hand side, how external, and potentially worldwide collaborators can contribute to and interact with the Open Data Hub team.

Internally, two distinct and separate environments exist: testing and production. The former is updated daily, while the latter only when the expected result (be it a new feature, a bug fix, or anything else) is ready to be published.

Both environments are updated with Continuous Integration using Jenkins, which monitors the git repositories and updates the environments.

External developers can push their own code to the git repositories (provided they have been granted with the permission to do so) and expect their work to be reviewed and tested by the Open Data Hub team.

5.5 Authentication in the Open Data Hub

The authentication layer is currently intended for **internal use only**, therefore it is **not** necessary to use authentication to access data provided by the Open Data Hub.

While the Open Data Hub project strives to offer only Open Data, it relies on third-party *Data Providers*, which may not offer the whole content of a dataset for public use. For this reason, an authentication mechanism has been implemented, which does however have no impact on users and on their use of the data.

Indeed, authentication in Open Data Hub is mainly used when exposing data to the consumer, which means by the Reader and in every single web service accessing the Reader, to allow the access to closed data in each dataset only to those who are allowed to, i.e., developers and members of the Open Data Hub team.

In the remainder of this section, we describe how authentication works within the Open Data Hub, because this information is of interest to users that might become app developers for the Open Data Hub project; further information about how to use authentication can be found in the *dedicated howto*.

There are currently two different authentication methods available:

- **Keycloak** is the default authentication server for all datasets that are accessed with the **API v2**.
- The **OAuth2 Authentication** follows the **RFC 6749** and is used for all the datasets in the mobility domain **when using the legacy API v1**.

Warning: The **Token-based Authentication** was used for the datasets in the tourism domain and is now not available anymore.

5.5.1 Keycloak for API v2

Keycloak Server is already used as authentication method for Open Data Hub's *internal infrastructure*. The same directions described in that section can be used.

5.5.2 OAuth2 Authentication

Warning: deprecated This authentication method is supported **only** for accessing datasets in the mobility domain with the **API v1**.

The OAuth2 authentication mechanism Authentication tokens are based on *JSON Web Token (JWT)* as defined in **RFC 7519#section-3**, to send *claims*.

For those not familiar with the OAuth2 mechanism, here is a quick description of the client-server interaction:

1. The client requests the permission to access restricted resources to the *authorisation server*.

2. The authorisation server replies with a **refresh token** and an **access token**. The access token contains an expire date.
3. The access token can now be used to access protected resources on the *resource server*. To be able to use the access token, add it as a Bearer token in the Authorization header of the HTTP call. **Bearer** is a means to use tokens in HTTP transactions. The complete specification can be found in [RFC 6750](#).
4. If the access token has expired, you'll get a HTTP 401 **Unauthorized** response. In this case you need to request a new access-token, passing your refresh token in the *Authorization* header as Bearer token. As an example, in Open Data Hub datasets Bearer tokens can be inserted in a **curl** call like follows:

```
~$ curl -X GET "$HTTP_URL_WITH_GET_PARAMETERS" -H "accept: */*" -H "Authorization: Bearer $TOKEN"
```

Here, `$HTTP_URL_WITH_GET_PARAMETERS` is the URL containing the API call and `$TOKEN` is the string of the token.

5.6 Authentication To Internal Infrastructure

Access to the Open Data Hub's internal infrastructure requires authentication, which is provided by **Keycloak**, an Open Source software that provides Identity and Access Management. In a nutshell, it acts as a broker to provide Single Sign On to different web sites and services; it also seamlessly interacts with Kerberos. More information and use cases can be found in the [official documentation](#).

Note: By *internal infrastructure* we mean also the access to Open Data that are already available to the Open Data Hub Team but not yet to the Data Consumers, and therefore require authentication.

Source code for both the authentication server and a few pre-cooked examples of applications configured to connect to it can be found in dedicated repository created by the Open Data Hub Team: the [authentication server](#).

5.6.1 Quick howto

Note: This howto describes the **old guidelines** to access authentication to Open Data Hubs internal infrastructure. A newer version of `odh-generic-client` has been implemented, that does not require anymore credentials and a **client_secret**. Therefore, all **curl** examples below can be safely used without all the corresponding options; credentials can be used for testing purposes, as explained in the [repository's README.md](#) file.

More information in the [dedicated repository](#).

In order to access the internal infrastructure, you need first to get a token, then use it together with the API. Both steps can be achieved using command-line tools, for a programmatic access to the date, which is the method shown here.

Request an access token

In order to receive an access token, you need in advance to have credentials for the Open Data Hub. If you do not have them, please open a ticket on issues.opendatahub.bz.it or send an email to . The same holds, if you plan to create an application that retrieves closed data from the Open Data Hub. For this, also other OAuth2 flows exist.

With your username and password, and a client secret (**my_username**, **my_password**, **the_client_secret**), the access token is granted to you with the following call:

Listing 5.1: Receiving an access topic

```
~$ curl -X POST -L "https://auth.opendatahub.bz.it/auth/realms/noi/protocol/openid-
↪connect/token" \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'username=my_username' \
--data-urlencode 'password=my_password' \
--data-urlencode 'client_id=odh-generic-client' \
--data-urlencode 'client_secret=the_client_secret'
```

Since the token expires after a given amount of time, it might prove necessary to refresh it, an action that can be done by replacing the parameters given in the query above with

Listing 5.2: Refreshing the access token

```
~$ curl -X POST -L "https://auth.opendatahub.bz.it/auth/realms/noi/protocol/openid-
↪connect/token" \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=refresh_token' \
--data-urlencode 'refresh_token=the_refresh_token' \
--data-urlencode 'client_id=odh-generic-client' \
--data-urlencode 'client_secret=the_client_secret'
```

Here, use the refresh token received from Listing 5.1.

Retrieve data with the token.

Once you received the access token, it is easy to use it in actual requests. The following API call shows how to get all **snames** and **mvalues** from the VMS dataset:

Listing 5.3: Retrieving data with the access token

```
~$ curl -X GET "https://mobility.api.opendatahub.bz.it/v2/flat/VMS/*/latest?select=sname,
↪mvalue" \
--header 'Content-Type: application/json' \
--header 'Authorization: bearer your-access-token'
```

Currently, data retrieved from the Open Data Hub are always open, except for some of the latest values and historical data: Only a subset of *m*-prefixed data from the `/latest` and `/from/to` API calls can be closed date. See section *Structure of the API calls and Payload* for more information about the API calls.

5.7 GITHUB Quick Documentation for Contributors

This section guides you in setting up on your local workstation the (forked) git repositories needed to contribute to the Open Data Hub project, along with some troubleshooting concerning pull requests and merge conflicts. For more detailed help, please refer to the online Github help, at <https://docs.github.com/en/>.

5.7.1 Prerequisites

In the following documentation some example names are used. Please replace them with your names:

- You need an account on Github to be able to fork projects and contribute to the Open Data Hub project.
- Replace `$USERNAME` with your username on GitHub.
- Replace `$BRANCH` with the branch name you will develop in your forked version.

5.7.2 Project Checkout

Before starting the development, you need to fork the original (upstream) repository.

1. Navigate to the repository on GitHub, e.g., <https://github.com/noi-techpark/bdp-core>.
2. Create a fork of the repository by clicking on the **Fork** button. If you are not logged in, you will be asked for a github username and password.
3. Navigate to your forked repository on GitHub, e.g., <https://github.com/\protect\T1\textdollarUSERNAME/bdp-core>.
4. Check out the forked repository on your local machine, using the link that appears in your repository (see [Figure 5.3](#)):

```
~$ git clone git@github.com:$USERNAME/bdp-core.git
```

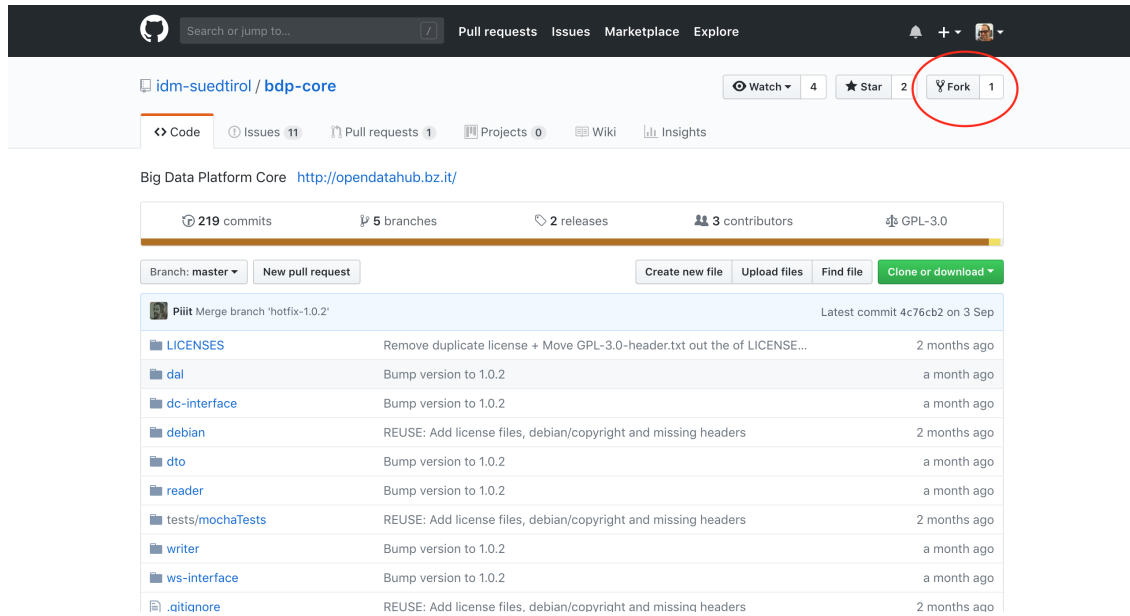


Figure 5.2: Fork the repository.

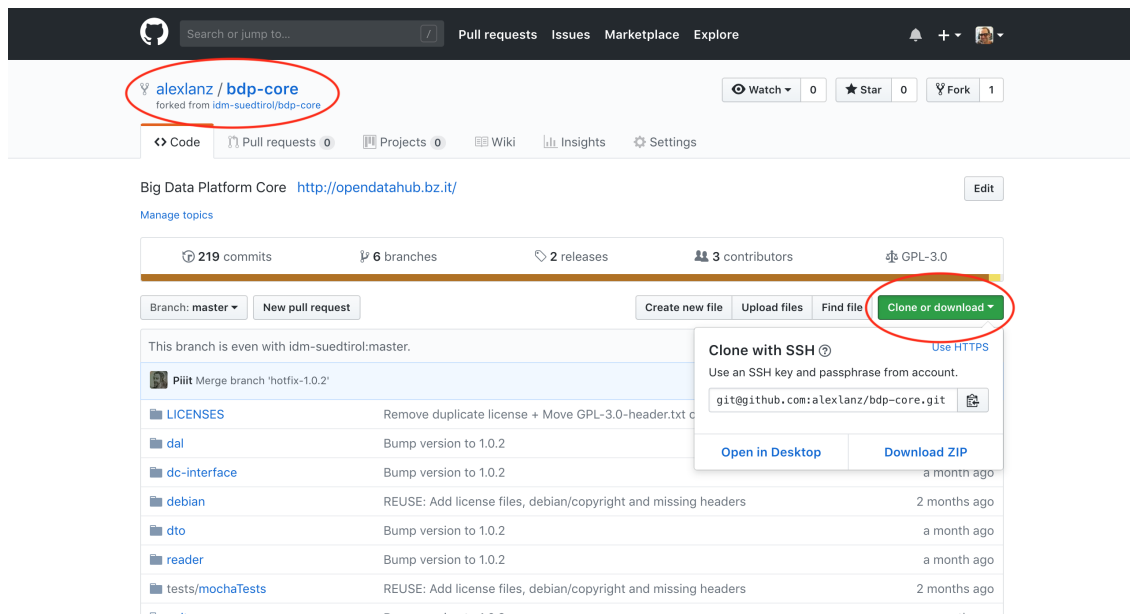


Figure 5.3: Clone the repository.

5.7.3 Create a pull request

In order to let your contribution be accepted in the Open Data Hub code base, you need to follow the following steps.

1. Checkout the **development** branch:

```
~$ git checkout development
```

2. Create a new branch from the **development** branch locally on your machine:

```
~$ git checkout -b test-branch
```

3. Make some changes to the code and commit them:

```
~$ git add -A
~$ git commit -m "Some commit message"
```

4. Push the new branch to GitHub:

```
~$ git push --set-upstream origin test-branch
```

5. Navigate to your feature branch on Github (<https://github.com/protect\T1\textdollarUSERNAME/bdp-core/pull/new/protect\T1\textdollarBRANCH>) to create a new pull request (see Figure 5.4).

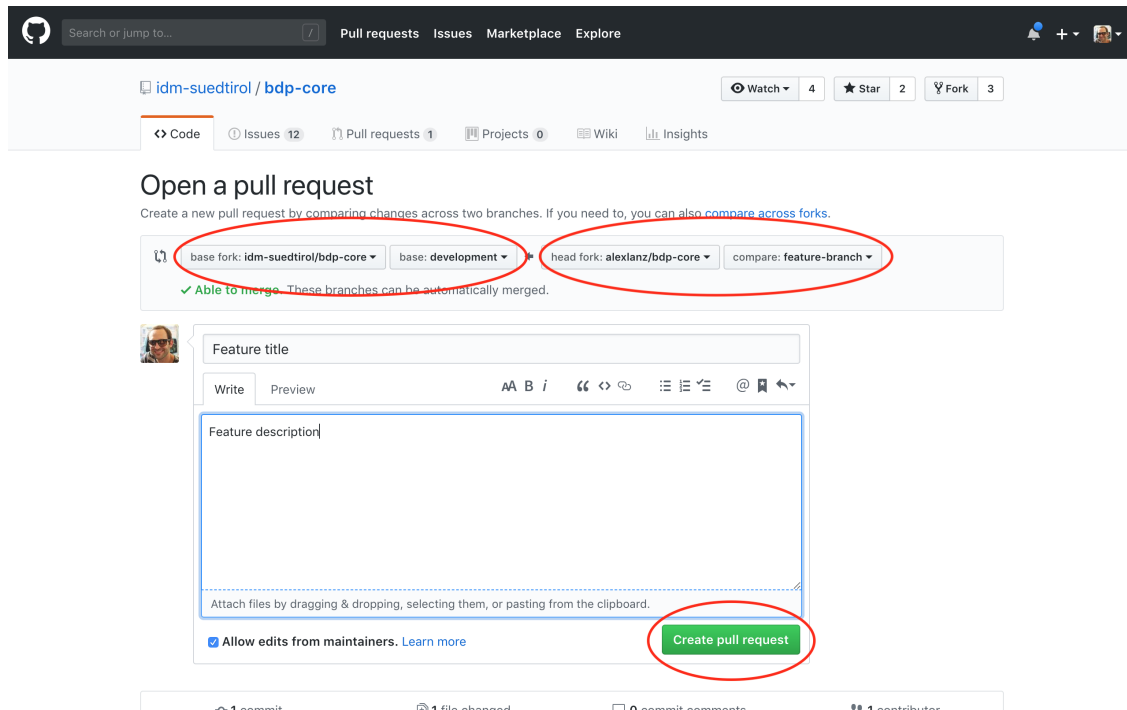


Figure 5.4: Create a pull request.

You can write some description as well, to describe your changes.

6. Commit and push any changes of the pull request to this new branch.
7. For every commit the continuous integration pipeline will execute the tests and display the results in the pull request, like shown in Figure 5.5
8. In addition, the detailed logs can be viewed under <https://ci.opendatahub.bz.it>.

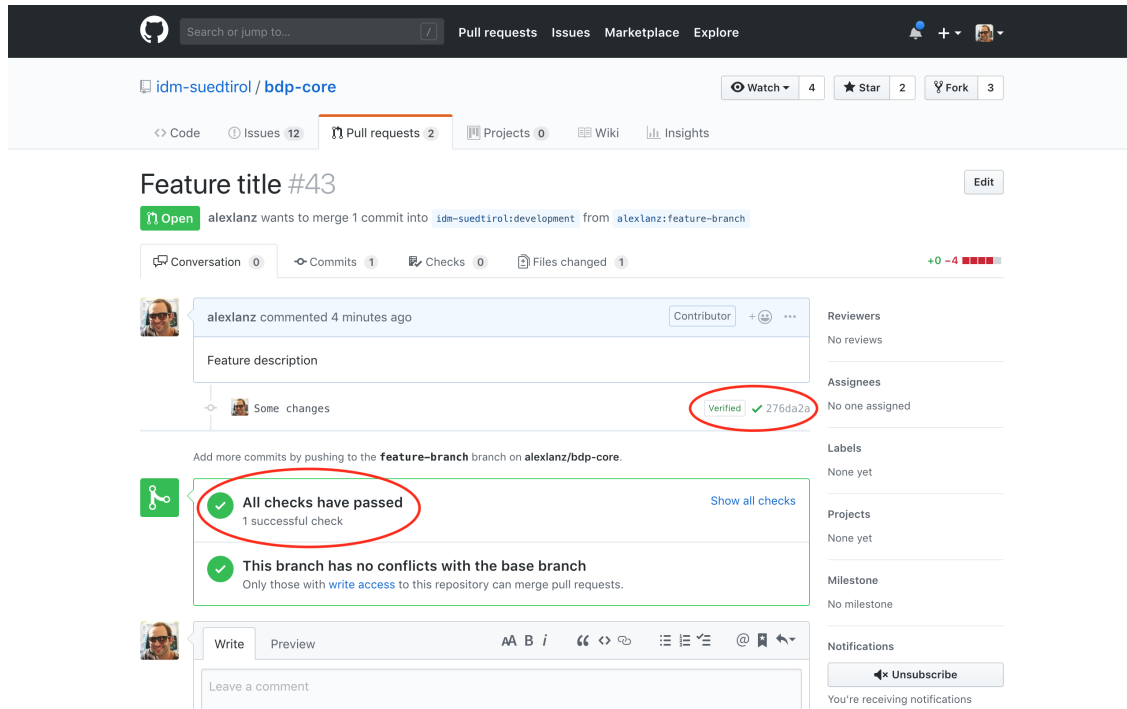


Figure 5.5: Show outcome of a pull request.

5.7.4 Syncing a Fork

Your forked repository does not receive the updates of the original repository automatically. To sync for example the **development** branch of the two repositories and to keep the forked repository up-to-date with all the latest changes of the **development** branch from the original repository, the following steps have to be performed.

Before you can sync your fork with the original repository (an upstream repository), you must configure a remote that points to the upstream repository in Git. A more detailed description for the following steps can be found in the [online Github help](#).

1. List the current configured remote repository for your fork.

```
~$ git remote -v
```

2. Specify a new remote upstream repository that will be synced with the fork.

```
~$ git remote add upstream https://github.com/noi-techpark/bdp-core.git
```

3. Verify the new upstream repository you've specified for your fork.

```
~$ git remote -v
```

You need sync a fork of a repository to keep it up-to-date with the original repository (upstream repository). A more detailed description for the following steps can be found in the online Github help <https://docs.github.com/en/github/collaborating-with-pull-requests/working-with-forks/syncing-a-fork>.

1. Fetch the branches and their respective commits from the upstream repository. Commits to **development** will be stored in a local branch, **upstream/development**

```
~$ git fetch upstream
```

2. Check out your fork's local **development** branch.

```
~$ git checkout development
```

3. Merge the changes from **upstream/development** into your local **development** branch. This brings your fork's development branch into sync with the upstream repository, without losing your local changes.

```
~$ git merge upstream/development
```

5.7.5 Resolving Merge Conflicts

When creating and working on a pull request, it could happen that the destination branch of the original repository will change. These changes could result in merge conflicts when pulling your code, like shown in Figure 5.6.

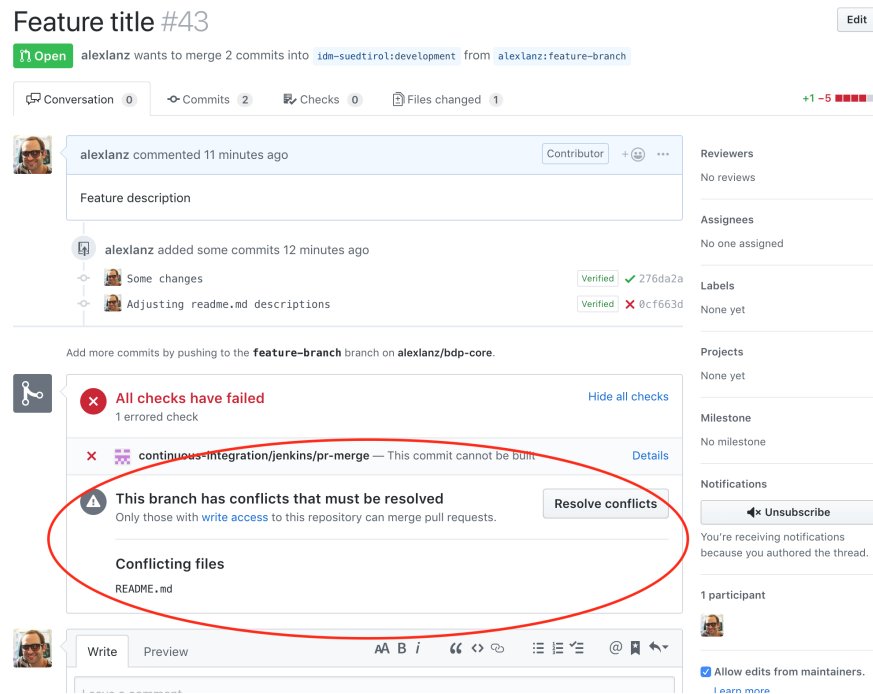


Figure 5.6: A Merge Conflict.

To resolve merge conflicts, the following steps must be performed.

1. *Sync your forked repository* and make sure your local destination (development) branch is up to date with the original (upstream) repository branch.
2. Check out your feature branch (replace *\$BRANCH* with the actual branch name).

```
~$ git checkout $BRANCH
```

3. Merge the changes of the development branch to the feature branch.

```
~$ git merge development
```

The command will output the files with merge conflicts. See sample output in [Figure 5.7](#).

```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Figure 5.7: Merge conflicts output.

4. Go to the listed files of the previous output and resolve all merge conflicts. The conflicts in the files begin with <<<<<< and end with >>>>>>. The ===== separates the two versions.

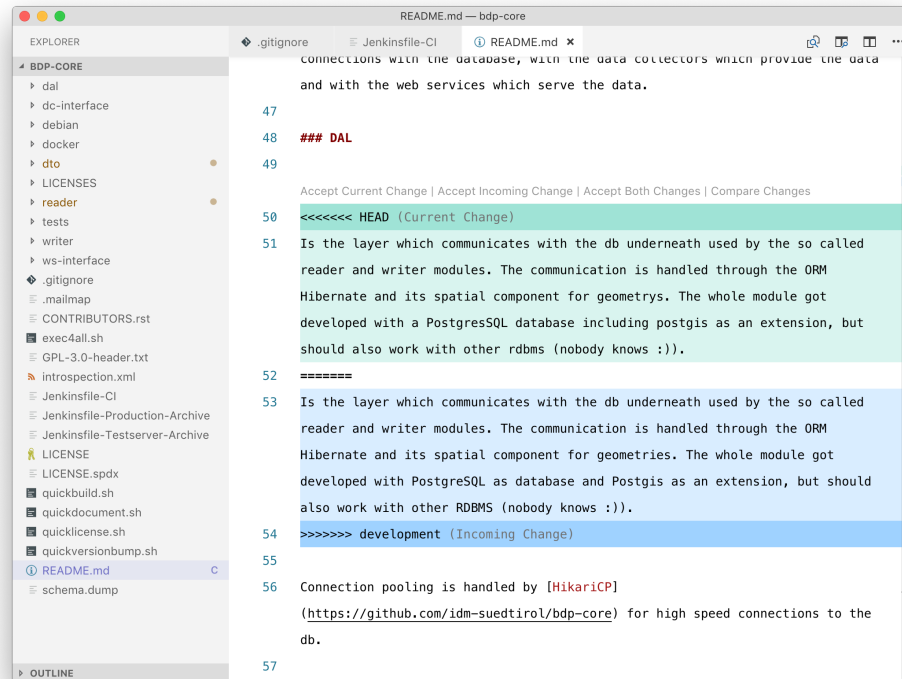


Figure 5.8: Solving a merge conflicts.

You can resolve a conflict by simply deleting one of the two versions of the code **and** the inserted helper lines beginning with <<<<<<, =====, and >>>>>>.

If none of the two versions is completely correct, then you can delete the conflict entirely and write your own code to solve the conflict.

5. Add all resolved files to the index, commit the changes and push the changes to the server.

```
~$ git add -A
~$ git commit
~$ git push
```

6. After resolving the merge conflicts, the pull request can be accepted.

A more detailed description can be found in the online Github help: <https://docs.github.com/en/github/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-using-the-command-line>

Feature title #43 Edit

Open alexlanz wants to merge 3 commits into `ide-suedtirol:development` from `alexlanz:feature-branch`

Conversation 0 Commits 3 Checks 0 Files changed 1 +1 -5

alexlanz commented 2 hours ago Contributor + ⌨ ...

Feature description

alexlanz added some commits 2 hours ago

- Some changes Verified ✓ 276da2a
- Adjusting readme.md descriptions Verified ✗ 0cf663d
- Merge branch 'development' into feature-branch Verified ✓ c5e757e

Add more commits by pushing to the **feature-branch** branch on `alexlanz/bdp-core`.

✓ **All checks have passed** Show all checks
1 successful check

✓ **This branch has no conflicts with the base branch**
Only those with [write access](#) to this repository can merge pull requests.

Write Preview AA B i “ < > ↺ ↻ ⋮ ⋮ ⋮ @ 🔍 ↶

Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Reviewers
No reviews

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Notifications
Unsubscribe
You're receiving notifications because you authored the thread.

1 participant



☒ Allow edits from maintainers. [Learn more](#)

Figure 5.9: A solved merge conflict.

APPS BUILT FROM OPEN DATA HUB DATASETS

This section features a list of applications—and their descriptions—built using the *Domains and Datasets* that the Open Data Hub team makes available.

Applications are grouped according to their status in three categories:

- Production status. These applications are already used in production environment.
- Development status, aka *beta stage*, shown by the  badge. Application in beta stage are developed actively and might already be suitable for a production environment.
- Experimental status, aka *alpha stage*, shown by the  badge. Applications falling in this category are in the early stage of development, and might be for example, a proof of concept or the outcome of a hackathon. They might not be maintained or developed further and should not be considered mature enough to be deployed in a production environment.

If you are developing one application with the data provided by datasets of the Open Data Hub project, send an email with a short description, an email contact and its status and we'll add it to the list.

6.1 Production Stage Apps

- <https://www.suedtirol.info/en> This portal is the official touristic site of the South Tyrol region. It uses the data extracted from the *Datasets in the Tourism Domain* to display events in the region of South Tyrol and other useful information to help tourists organise their holiday in South Tyrol. All information are available also for every region that composes South Tyrol, including historical remark and proposed taste itinerary.

Additionally, it is also possible to subscribe to the newsletter, to browse or search for events and points of interest. For most of the points of interest there are suggestions for hiking tracks or walks to reach them.

Hotel description and availability with options for booking are additional services offered by the site.

- South Tyrol Guide, the official smartphone app for exploring and experiencing South Tyrol, available for both [Android](#) and [iPhone](#) mobile devices. This app offers all the functionalities of the above-mentioned web site, plus the ability to locate events, points of interest and the like near your current position.

6.2 Beta Stage Apps

- <https://mobility.meran.eu/> This web site is the first example of a Mobility-as-a-Service application; it includes real-time information of multiple mobility services, like public transportation, places of interests, car sharing services, parking lots, and more in the city of Merano/Meran.
- <https://mobility.bz.it/> This web site is the counterpart of the previous one for the city of Bolzano/Bozen.
- <https://parking.bz.it/> A web site that displays the real-time parking availability of off-street parking lots in South Tyrol. On mobile devices, it can also show directions from your current position to the chosen parking lot.
- <https://traffic.bz.it/> Some streets in South Tyrol are monitored for real-time vehicular travel times; the data collected are used by this web site to show traffic slowdowns or jams.
- <https://bus.bz.it/> This web site shows the real-time positions of the buses managed by the public transport operator SASA. Urban or suburban bus lines can be shown, and for each bus can be shown the next few stops and an estimate of the arrival time.
- <https://map.clean-roads.eu/> One of the CLEAN-ROADS project outcomes, this web site shows real-time data of the meteorological stations that are situated along public streets.

6.3 Alpha Stage Apps

All the projects listed in this section have been built during various hackathons and must be considered as *Experimental*.



Year 2018

Summer Lido Hackathon

Projects developed during the Summer Lido Hackathon 2018

- <https://hackathon.bz.it/project/south-tyrol-crime-scene-stcs-> is an interactive thriller combining traditional storytelling and augmented reality.
- <https://hackathon.bz.it/project/sama-smart-application-medical-appointment> is a prototype for the booking of appointments in the South Tyrolean hospitals.
- <https://hackathon.bz.it/project/ifc-converter-and-aivrtour> is on the one side a converter of 3D reality data formats (from ifc to dae/obj), while on the other side it applies AI to virtual reality for museums, real estates, and on tourism.
- <https://hackathon.bz.it/project/travel-win> is an app for South Tyrol tourists that can collect points to receive gifts.
- <https://hackathon.bz.it/project/ugo> Get your pictures printed on paper and delivered to your hotel.
- <https://hackathon.bz.it/project/sportmap.net> Creation of a OpenData map with trendy sport locations, courses, & events.
- <https://hackathon.bz.it/project/game-of-alps> Get unique experiences by completing challenges around South Tyrol. Gather crystals and collect rewards from local tourist offices. Additionally get discounts for restaurants and entrance tickets.

HackTheAlps

Projects developed during [HackTheAlps 2018](#).

- [Activity Crystal](#) goal of the project was to build something simple & fun that gets people off their couches.
- [Explore South Tyrol](#) aims at providing the user with a new experience when visiting South Tyrol. The user can discover and share cool new places in South Tyrol in a truly immersive way.
- [Offtrack](#) is an app that bridges ski rentals, skipass sellers, and insurances, to create a new kind of business, centred on the safety of ski enthusiasts.

Vertical Innovation Hackathon

Projects developed during [Vertical Innovation 2018](#).

- [Smart Bivouac](#) is the project that won the **Cassa di Risparmio / Sparkasse award** at the Hackathon. The idea on which the project is based is to avoid crowded bivouacs in the South Tyrolean area with the use of LoRa technology, that helps in booking a place in a bivouac.
- [RESK](#) is the project that won the **Wuerth Phoenix award** at the Hackathon, which had the use of Open Data Hub data in its criteria. The goal of the project is to reduce the first intervention times, for EMT and first-aider to react more promptly to help requests and save more lives.
- [Back forth](#) has the goal to solve the commuting problems in South Tyrol with an eye on eco-friendliness and Open Source .
- [Pool me](#) aims at promoting carpooling in South Tyrol by logging and storing all rtavels done.
- [WideOpen](#) is a change management platform that helps companies in adopting flexible working hours policies in order to make the rush hour a relic of the past.

Moreover, the following applications use the Open Data Hub as a feature, but not as their core.

- <https://hackathon.bz.it/project/the-smart-team>
- <https://hackathon.bz.it/project/green-revolution>
- <https://hackathon.bz.it/project/my-south-tyrol>
- <https://hackathon.bz.it/project/think>
- <https://hackathon.bz.it/project/webike>

Year 2019

NOI Hackathon Summer edition

Projects developed during the [Summer Lido Hackathon 2019](#)

- [WeMap](#) Lit element component for display opendata on google maps.
- [Tito](#) is an app and web component that analyses historical data about trips made in South Tyrol and suggests the best one to allow people to share them and plan to travel together along them.

NOI Hackathon SFScon Edition

Projects developed during NOI Hackathon SFScon Edition 2019.

- [Greenify](#) is a Web Component and user interface intended for tourists and hotels interested in sustainable environment. The authors strive for South Tyrol to become an environmental friendly region, therefore this Web Component classify hotels and assign them scores and certifications of their environmental friendliness.
- [The Vecia](#) follows Jane's journey in South Tyrol and shows a web component to enhance sustainable tourism.
- [Sharing is Caring](#) consists of a web component that allows tourist to access easily information about public transportation in South Tyrol and allow them to move around without using their own car.
- [Green Speed](#) gather information from different sources to help travelers in South Tyrol to get directions using public transportation and provide them with the actual carbon footprint.

APPENDICES

Changed in version 2022.03: Moved Open Data Hub Architecture in the appendices.

The following is the list of appendices.

7.1 Open Data Hub Architecture

The architecture of the Open Data Hub is depicted in [Figure 7.1](#), which shows its composing elements together with its main goal: To gather data from **Data Sources** and make them available to **Data Consumers**, which are usually third-party applications that use those data in any way that they deem useful, including (but not limited to) study the evolution of historical data, or carry out data analysis to produce *Statistical Graphics*.

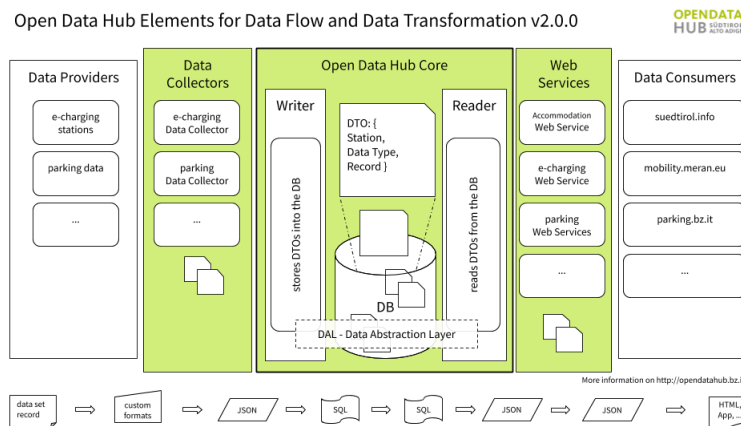


Figure 7.1: The Open Data Hub architecture with the components (top) and the data formats used (bottom) during each data transformation.

At the core of the Open Data Hub lays **bdp-core**, a java application which contains all the business logic and handles all the connections with the underlying database using the DAL (Data Access Layer). The Open Data Hub Core is composed by different modules: A **Writer**, that receives data from the Data Sources and stores them in the Database using the DAL and a **Reader** that extracts data form the databases and exposes them to Data Consumers using APIs on REST endpoints.

Communication with the Data Sources is guaranteed by the **Data Collectors**, which are Java applications built on top of the **dc-interface** that use a DTO (Data Transfer Object) for each different source to correctly import the data. Dual to the **dc-interface**, the **ws-interface** allows the export of DTOs to web services, that expose them to **Data Consumers**.

The bottom part of [Figure 7.1](#) shows the *Data Format* used in the various steps of the data flow. Since the data are exposed in JSON, it is possible to develop applications in any language that uses them.

Records in the Data Sources can be stored in any format and are converted into JSON as DTOs. They are then transmitted to the Writer, who converts them and stores them in the Database using SQL. To expose data, the Reader queries the DB using SQL, transforms them in JSON's DTOs to the Web Services who serve the JSON to the Data Consumers.

7.1.1 The Elements of the Open Data Hub in Details

As [Figure 7.1](#) shows, the Open Data Hub is composed by a number of elements, described in the remainder of this section in the same order as they appear in the picture.

Data Providers

A *Data Provider* is a person, company or public body that supplies to the Open Data Hub some data or dataset, which usually belongs to a single domain. Data are automatically picked up by sensors and stored under the responsibility of the Data Provider in some standard format, like for example CSV or *JSON*.

Note: Since a data provider may decide at some point to not publish its data on the Open Data Hub anymore, or new data providers can join the Open Data Hub in the future, they are not an official part of the Open Data Hub. You can learn more on this, including the current list of data providers, in the *dedicated section* of the documentation.

Dataset

A dataset is a collection of records that typically originate from one Data Provider, although, within the Open Data Hub, a dataset can be built from data gathered from multiple Data Providers. The underlying data format of a dataset **never** changes.

Data Collector

Data collectors form a library of Java classes used to transform data gathered from Data Providers into a format that can be understood, used, and stored by the Open Data Hub Core. As a rule of thumb, one Data Collector is used for one Dataset and uses DTOs to transfer them to the Open Data Hub Core. They are usually created by extending the **dc-interface** in the bpd-core repository.

DTO

The Data Transfer Object are used to translate the data format used by the Data Providers, to a format that the Writer can understand and use to transfer the data in the Big Data infrastructure. The same DTO is later used by the Reader (see below) to present data. DTOs are written in **JSON**, and are composed of three *Entities*: **Station**, **Data Type**, and **Record**.

Writer

With the Writer, we enter in the Open Data Hub Core. The Writer's purpose is to receive DTOs from the Data Collectors and store them into the DB and therefore implements all methods needed to read the DTO's *JSON* format and to write to the database using SQL.

ODH Core

The Open Data Hub Core lays at the very core of the Open Data Hub. Its main task is to keep the database updated, to be able to always serve up-to-date data. To do so, it relies on the Writer, to gather new or updated data from the data collectors and keeps a history of all data he ever received. It also relies on the Reader to expose data to the data consumers. Internal communication uses only SQL commands.

DAL

The Data Abstraction Layer is used by both the Writer and the Reader to access the Database and exchange DTOs and relies on Java Hibernate. It contains classes that map the content of a DTO to corresponding database tables.

Database (DB)

The database represents the persistence layer and contains all the data sent by the Writer. Its configuration requires that two users be defined, one with full permissions granted -used by the writer, and one with read-only permissions, used by the Reader.

Reader

The reader is the last component of the Core. It uses the DAL to retrieve DTOs from the DB and to transmit them to the web services.

Web Services

The Web Services, which extend the **ws-interface** in the Open Data Hub Core repository, receive data from the Reader and make them available to Data Consumers by exposing APIs and REST endpoints. They transform the DTO they get into JSON.

Data Consumers

Data consumers are applications that use the JSON produced by web services and manipulates them to produce a useful output for the final user. As mentioned in the section *Project Overview*, *application* is intended in a broad sense: it can be a web site, a software application for any devices, a communication channel, or any means to use the data.

Also part of the architecture, but not pictured in the diagram, is the `persistence.xml` file, which contains the credentials and postgres configuration used by both the Reader and Writer.

7.2 Glossary

API

The Application Programming Interface is a collection of methods that a software program makes available to allow interaction with other programs.

Claim

In JSON Web Token, a claim is a piece of information about a subject, structured as a key/value pair.

CSV

CSV stands for Comma Separated Value, and is a file in which the content is organised in a fixed number of fields per line, separated by a comma (,) or some other symbol, like a semi-colon, a slash, or a vertical bar.

DAL

The DAL is used by the reader and writer to communicate with the database. See the *detailed description*.

Data Collector

A component of the Open Data Hub, a data collector is used to gather data from datasets and send them to the Open Data Hub. See the *detailed description*.

Data Consumers

Applications that use data received from the Web Services. See the *detailed description*.

Data Format

Data format is the way information is encoded and exchanged between applications.

Data Provider

A Data Provider is an entity that supplies data or datasets to the Open Data Hub. See how it *is integrated* in the Open Data Hub and a *detailed description*.

Database

Also known as persistence layer, the database (“DB”) stores all the data received by the writer. See the *detailed description*.

Dataset

A dataset is a collection of records from a Data source. See the *detailed description*.

Domain

A domain is a category of interest to which one or more datasets belong to.

DTO

A core component of the Open Data Hub, the DTO transforms the data format of a Source into a Open Data Hub-understandable format. See the [detailed description](#).

Endpoint

An Endpoint is an online resource that make data available, usually through REST APIs.

JSON

The JavaScript Object Notation is a lightweight data format to ease the exchange of data between computer and its understanding for humans. Essentially a JSON file is a sequence of key-value pairs, organised into lists (arrays, sequences, vectors). Nesting of key-values and of lists is supported.

JSON Web Token

It is a mechanism to exchange a claim between two parties, used for authentication purposes when the claim is digitally signed and/or encrypted.

Key-value

Also called name-value pair or attribute-name pair, a key-value pair is a simple data structure in which information are stored as tuples {attribute, value}, with no constraint of uniqueness on both attribute and value.

ODHtags

In the tourism domain, this name refers to all the tags/filter that refer to data that have been validated by the Open Data Hub team.

Persistence Layer

Another name for Database, see the above entry or the [detailed description](#).

Reader

A core component of the Open Data Hub, the Reader extract data form the Database and sends it to the web services. See the [detailed description](#).

REST API

RESTful API

A REST(ful) API is a Web Service that adheres the architectural constraint of a [RESTful system](#). It is composed of a URI, a collection of methods to interact with the resources offered by the Web Service, and a media type defining the accepted data formats.

Sensor

Within the Open Data Hub, a sensor is intended as a kind of *device* that gathers data and sends them to another device which stores them in a machine-readable format, used to exchange or publish them. Depending on the domain a sensor may collect environmental data in the mobility domain (like, e.g., temperature, humidity, pressure), but in the tourism domain a *sensor* can collect the guests in a hotel or the people attending at an event. In these cases, the *device* is usually a human (e.g., the hotel's receptionist and the organiser of the event), and the data are digitalised manually.

Statistical Graphics

Statistical graphics are means to display statistical data with the purpose to ease their interpretations. Common statistical graphics include pie charts, histograms, and scatter plot.

Web Component

A Web Component is a set of API that allows the creation of interoperable HTML widgets that can be shared and reused.

Web Services

In the context of the Open Data Hub Project, web services expose to Data Consumers the data received from the reader. See the [detailed description](#).


Writer

The Writer is a core component of the Open Data Hub. It receives data from the Data Collectors and stores them in the Database. See the [detailed description](#).

7.3 Licenses and TOS for the Open Data Hub material

The resources that are part of the Open Data Hub Project are subject to different licenses, which are described in section [Licenses for Open Data Hub resources](#). Derivative material built using Open Data Hub material is also subjected to different licenses, depending on its purpose, as shown in [Figure 7.2](#).

Free Libre Open Source Software License Matrix v1.0



	Network copyleft All derivative works provide the 4 freedoms. Even to users who just use the program (webapp users).	Copyleft All derivative works provide the 4 freedoms.	File/weak copyleft Derivative works provide the 4 freedoms, for the files included in the original work. Not necessarily for other files in the derivative work.	Non-copyleft Derivative works do not guarantee to provide the 4 freedoms anymore.
For Web Apps (SaaS) Guarantees freedoms to users and developers .	AGPLv3			
For Apps Guarantees freedoms to developers .		GPLv3		
For Libraries Can be used by proprietary (non open) projects.			LGPLv3 or MPLv2	
For Frameworks Can become proprietary (non open) projects.				APLv2

AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.en.html>
 GPLv3 <https://www.gnu.org/licenses/gpl-3.0.en.html>
 LGPLv3 <https://www.gnu.org/licenses/lgpl.html>
 MPLv2 <https://opensource.org/licenses/MPL-2.0>
 APLv2 <https://opensource.org/licenses/Apache-2.0>

More information on <http://opendatahub.bz.it>

Figure 7.2: Licenses for the Open Data Hub and derivative material.

7.3.1 The FLOSS four freedoms

The *four essential freedoms* are the four basic principle to which a software program must comply to be defined free software. As stated on the [What is free software?](#) web page (on which you can find a lot more information and details), they are:

- The freedom to run the program as you wish, for any purpose (**freedom 0**).
- The freedom to study how the program works, and change it so it does your computing as you wish (**freedom 1**).. Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help others (**freedom 2**).
- The freedom to distribute copies of your modified versions to others (**freedom 3**). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

7.3.2 Licenses for Open Data Hub resources

The Open Data Hub Project processes dataset, possibly supplied by third-party sources (i.e., *Data Providers*), which may contain closed data; however, only Open Data are returned to the users' queries.

According to the main goal of the Open Data Hub Project, we have defined licenses for its different components and we use badges across the documentation for a better visibility. As a rule of thumb, we try to do our best to deliver **Open Data**, by developing **Free/Open Source software** that is publicly available on github, and by using an **Open Standard** for the API used to access data.

These licenses are applied to the Open Data Hub components:

- All the **software** released within the Open Data Hub is Free software and complies with the GPLv3 license.
code **GPLv3** Code repositories can be found at <https://github.com/noi-techpark>.
- The **Datasets** currently expose only Open Data that are in the public domain, so they are released as CC0.
dataset **CC0**
- The **APIs** have no license yet, since we are in the process to define which among the CC licenses could fit best. See [Figure 7.3](#) for an overview and quick description of CC licenses and derivative material.


CC0 Licensed Data

Open Data Hub provides a live updated table about the number of CC0 licensed data it contains. Please note, this data is calculated on some datasets and does not consider all datasets of the whole Open Data Hub yet.

<https://databrowser.opendatahub.bz.it/Home/LicenseStatus>

Note: There is an additional clarification about the licence for any content that is retrieved from the datasets in JSON format, which is detailed in [Section License of the JSON Responses](#).

Data and Content License Matrix v1.0



	Zero No limitations.	Attribution Licensees may copy, distribute, display and perform the work and make derivative works and remixes based on it only if they give the author or licensor the credits (attribution) in the manner specified by these.	Share-alike Licensees may distribute derivative works only under a license identical ("not more restrictive") to the license that governs the original work.	Non-open License imposes restrictions in sharing, remixing or other use.
Easy to be used by third parties. Just use it without any effort.	CC0			
Third parties have to include attribution.		CC-BY		
Third parties have to include attribution and cannot apply restrictions.			CC-BY-SA	
No permission granted. Or at least no transparent permission granted.				Other or no license

CC0 <https://creativecommons.org/publicdomain/zero/1.0/>
CC-BY <https://creativecommons.org/licenses/by/2.0/>
CC-BY-SA <https://creativecommons.org/licenses/by-sa/2.0/>

More information on <http://opendatahub.bz.it>

Figure 7.3: Creative Common Licenses and derivative material.

License of the JSON Responses

Whenever you query the data in the Open Data Hub, the snippet that you retrieve always includes a block of information called `LicenseInfo`, similar to the following one:

```

1 {
2   "LicenseInfo": {
3     "Author": "",
4     "License": "CC0",
5     "ClosedData": false,

```

(continues on next page)

(continued from previous page)

```

6     "LicenseHolder": "https://www.lts.it"
7   }
8 }

```

The highlighted line shows a licence, which in this case is **CC0**, i.e., public domain and therefore freely reusable.

This block is always included as a child node within a JSON record that starts with an ID and a number of additional information, which may include also hyperlinks to resources that are external to the Open Data Hub, like in the following snippet—shortened for the sake of simplicity, which refers to a webcam and contains a link to an external provider where to find actual resources (e.g., streams and images) from that webcam.

```

1 {
2   "Id": "D3659E1F111C4CDB2EC19F8FC95118B7",
3   "Active": true,
4   "Streamurl": null,
5   "Webcamurl": "https://webtv.feratel.com/webtv/?pg=5EB12424-7C2D-428A-BEFF-
6   ↪0C9140CD772F&design=v3&cam=6323&c1=0",
7   "LicenseInfo": {
8     "Author": "",
9     "License": "CC0",
10    "ClosedData": false,
11    "LicenseHolder": "https://www.lts.it"
12  }
13 }

```

Whenever hyperlinks like the one shown in line 5 above appear, it must not be implied that the license mentioned in the LicenseInfo block (again, CC0) is applied to them: everything contained in that link may be covered by a different licence.

Indeed, the **License** mentioned in LicenseInfo nodes refer only to content of the parent node—i.e., the one that starts with “Id”, not to the content of any of the other children nodes, including Streamurl and Webcamurl.

7.3.3 APIs Terms of Service

The Open Data Hub project is already used in production for NOI internal projects, and in particular it is the data hub used by the South Tyrolean tourism portal, <https://www.suedtirol.info/en/>.

The public API are in early development and therefore should be still considered as a **beta** version. If any third party would like to use a stable version of the APIs in its production environment, a special agreement must be signed with NOI [techpark](#). You can contact help@opendatahub.bz.it for any information.

7.4 Changelog

Available Changelogs:

- [2021.03](#)
- [2021.04](#)
- [2021.05](#)
- [2021.06](#)

- [2021.08](#)
- [2021.09](#)
- [2021.12](#)
- [2022.03](#)
- [2022.03](#)

This section contains the Changelog of the Open Data Hub project’s documentation.

- Previous to June 2020, we maintained no changelog.
- The list of changes made from June, 2020 to February, 2021 can be [downloaded](#) as a text file.
- Since March 2021 we adopted a more structured approach: This section indeed will contain an entry for each change made to the documentation.

Note: We do not schedule regular releases of the documentation, we rather publish new versions “*When they are ready*”, therefore each *version* is identified by a string **YYYY.MM**, in which *YYYY* is the year and *MM* the month of the release.

7.4.1 2021.03

Released: 31 March 2021

- **[Improvement]** ¶ Add **changelog** extension.
References: [#194](#)
- **[Improvement]** ¶ Add AlpineBits to the *Accessing the Open Data Hub* section.
References: [#203](#)
- **[Bugfix]** ¶ Fix layout of datasets in *Other Domains*.
References: [#199](#)

7.4.2 2021.04

Released: 30 April 2021

- **[Improvement]** ¶ Add to each dataset a direct permalink that can be copied and sent to third party.
References: [#206](#)
- **[Improvement]** ¶
 - Replace tabs in the howto list with subsections.
 - Remove the mobility howto for deprecated API v1.
 - Reduce size of images in howtos and align them if layout allows.References: [#208](#)
- **[Bugfix]** ¶ Add correct images to *How to Access Analytics Data in the Mobility Domain* howto.
References: [#208](#)
- **[Bugfix]** ¶

- Remove drop downs from all the lists of *datasets* (Mobility, Tourism, Other)
- fix the panel's width to avoid scrollbars whenever possible.

References: #212

- **[Bugfix]** ¶ Add troubleshooting section to *R howto*.

References: #213

- **[New Feature]** ¶ New Howto: *How to Access Open Data Hub Data With R and SPARQL*

References: #204

7.4.3 2021.05

Released: 31 May 2021

- **[Improvement]** ¶ A lot of improvements have been added to the general structure of the documentation, the most important being:
 - reorder ToC and make some section more prominent
 - made bash code snippets more usable
 - made accessing methods more immediate to see

For more details, please check the reference.

References: #209

- **[Improvement]** ¶ The technical content of the *getting started* howtos has been moved to the *Datasets* section, making them shorter. Also a few examples have been added to *How to Access Mobility Data With API v2*.

References: #214

7.4.4 2021.06

Released: 30 June 2021

- **[Change] [Improvement]** ¶ The Tourism domain was modified and improved in several points, which are reflected in the documentation:
 - New Swagger and API URLs
 - Localised methods have been definitely removed
 - A new *extlink* to shorten URLs of tourism API in the documentation source code has been introduced
 - The *How to access Tourism Data?* article has been modified to include the API browsable interface
 - Tourism datasets have been ordered lexicographically

References: #220

- **[Change] [Improvement]** ¶ The description of the new Knowledge Graph underlying the Mobility domain has been added. Also the *SPARQL Howto* has been updated to reflect the new precooked queries and layout.

References: #219

7.4.5 2021.08

Released: 31 August 2021

- **[Improvement] [Bugfix]** ¶ Update old URLs, fix broken links.
References: #227
- **[Improvement]** ¶ Add statistics about Tourism's Open Data and CC0-licensed images
References: #232
- **[Improvement]** ¶ Rearrange subsections in sections *Accessing the Open Data Hub* and *Domains and Datasets*, add a new FAQ entry and reformat FAQ section.
References: #221
- **[Improvement]** ¶ The `sphinx-panels` extension has been replaced by its successor, `sphinx-design`.
References: #233
- **[Change] [Improvement]** ¶ Keycloak has been introduced as default authentication method for the Open Data Hub.
References: #223
- **[Change]** ¶ The URL of the Tourism API has been updated
References: #231
- **[New Feature]** ¶ New filters for the tourism domain: `rawfilter`, `rawsort`, and `removenullvalues`.
References: #224

7.4.6 2021.09

Released: 30 September 2021

- **[Improvement] [New Feature]** ¶ AlpineBits DestinationData and HotelData have now a dedicated page.
References: #225
- **[Bugfix]** ¶ Fix requirements.txt file.
References: #244

7.4.7 2021.12

Released: 31 December 2021

- **[New Feature]** ¶ Add WeatherHistory dataset.
References: #247

7.4.8 2022.03

Released: 31 March 2022

- **[New Feature]** ¶ Add section *Quickstart*.

References: #250

7.4.9 2022.03

Released: 31 March 2022

- **[Improvement]** ¶ Major changes to sectioning: unify sections *Project Overview*, *Getting Involved*, and *Accessing the Open Data Hub*; move description of domains to *Domains and Datasets* section, move *Open Data Hub Architecture* to *Appendices*.

References: #251

- **[New Feature]** ¶ Add section *Quickstart*.

References: #250

FREQUENTLY ASKED QUESTIONS

This section contains answers to questions frequently asked by people who want to contribute to the Open Data Hub project or search for information about the project.

Q: What is this project about?

A: The project is described in section *Introduction*.

Q: I am interested in taking part in the Open Data Hub project.

A: Check section *Getting Involved*.

Q: I am a developer, are there guidelines for the development?

A: Sure! Check the dedicated section: *Resources for Developers*.

Q: How do I access the data served by the Open Data Hub?

A: You can see if section *List of HOWTOs* contains what you are looking for. If not, you can open an issue in our bug tracker.

Q: The project misses a... [feature, dataset, howto, etc.]!

A: Please check section *Bug Reporting and Feature Requests* then open an issue on one of our bug trackers.

Q: What's the licence of Open Data Hub's data?

A: All the data retrieved from the Open Data Hub is distributed with an Open license. Please check sections *Datasets*, *Open Data, and Licenses* and *License of the JSON Responses* for detailed information.

Symbols

-H
command line option, 60
-X
command line option, 60
--header
command line option, 60
10-07-2018, 89

A

API, 107

C

Claim, 107
command line option
-H, 60
-X, 60
--header, 60
CSV, 107

D

DAL, 107
Data Collector, 107
Data Consumers, 107
Data Format, 107
Data Provider, 107
Database, 107
Dataset, 107
Date, 89
date, 79, 89, 90
Date.prototype.toISOString(), 90
de_DE, 89
Domain, 107
DTO, 108

E

en_US, 78, 88
Endpoint, 108
environment variable
10-07-2018, 89
Date, 89
date, 79, 89, 90

Date.prototype.toISOString(), 90
de_DE, 89
en_US, 78, 88
fetchsize, 85
it_IT, 89
localhost, 63
money, 89
numeric, 89
timestamp, 89, 90
timestamp with timezone, 79
timestamp without time zone, 89
UTF8, 78, 88

F

fetchsize, 85

I

it_IT, 89

J

JSON, 108
JSON Web Token, 108

K

Key-value, 108

L

localhost, 63

M

money, 89

N

numeric, 89

O

ODHtags, 108

P

Persistence Layer, 108

R

Reader, [108](#)

REST API, [108](#)

RESTful API, [108](#)

RFC

RFC 6749, [91](#)

RFC 6750, [92](#)

RFC 7519#section-3, [91](#)

S

Sensor, [108](#)

Statistical Graphics, [108](#)

T

timestamp, [89](#), [90](#)

timestamp with timezone, [79](#)

timestamp without time zone, [89](#)

U

UTF8, [78](#), [88](#)

W

Web Component, [108](#)

Web Services, [108](#)

Writer, [108](#)