REST API v1

Quickstart Guide


UNDER
CONSTRUCTION

Content

1 Preface

Cargoloy is a very young project and still under construction.
We are currently looking for partners, angels, fundings and the like and all you can see here is the work of a highly motivated team.

However, most things you will read on the upcoming pages have already proven their fitness, steadyness and have undergone serious tests.
You might have played around with the demo on https://www.cargoloy.com/demo already, then probably contacted your CTO and tickled out a thin time slot for exploring things more deeply. Especially the major question of how to integrate Cargoloy into your own infrastructure.

This paper is about Cargoloy's REST-API.

It does not cover the libraries or how to integrate them into foreign Java infrastructures, which is possible and much easier than boiling eggs.

Showcase

From Moscow to Madrid

Blue:	Streets and Sea
Green:	Streets only
Red:	Streets and Air

2 Restrictions

Our API is not free. As an anonymous user you are restricted with regards to the requests you may fire within a given time.

For higher rates you'll need an x-api-key.

2.1 x-api-key

Adding a registered key to your request opens the door to a faster world. This key must be sent in the header of an http request and is called **x-api-key**.
It is a secret key, so do not share it with other people, unless you are going to slow down your bandwith. This also applies to parallel request, fired from one single client.

Please contact us directly, if you need one.

3 First contact

Take the following link to check if things are working on your local machine.
You can paste it into the address field of your Browser or, if you prefer command line tools, use curl or one of its relatives. Or just click:

https://api.cargoloy.com/v1/route/@1/@2

Hamburg to Miami (both Ports)

Recalling the URL-Parameters above, we should be able to find them somewhere in the GeoJson-response. Just follow this path to find our source parameter @1:

`features[0].properties.leg.source.poi.id`

It is a POI (Point of Interest) and a port. Ports are hubs, which connect different types of network.

Here is the result, already formatted by Firefox:


```
type:                    "FeatureCollection"
▼features:
  ▼0:
    type:                "Feature"
    ▼geometry:
      type:              "MultiLineString"
      ▶coordinates:      […]
    ▼properties:
      ▼leg:
        gid:             1870089819
        version:         "5.2.105"
        api:             1
        type:            "SEA"
        ▼source:
          locator:       "poi"
          ▶vertex:       {…}
          ▼poi:
            id:          1
            idx:         5449
            clazz:       10
            lat:         53.532238
            lon:         9.9072838
            name:        "HAMBURG (Port) DE"
            nearby:      false
        ▼target:
```

4 Basic Concept

4.1 Location Types

A graph denotes the routable data. It is the network composed of one huge collection of ways (the edges) and crossings (the vertices). Vertices are the glue. They connect edges at intersections and allow to hop between multiple network types, e.g. Street, Sea, River, Air or Rail.
Each edge or vertex has a unique index. But this is only a technical pointer and hence will change with each new graph version or build. An index can be used in sessions to retrieve information more quickly but its transient nature makes it useless for any kind of persistence.
What we need are reliable locations, something we can address even when things change. And there is only one address we can trust, the geographic position.
However, we must be able to handle known places, deal with random positions and of course the points we need for the routing.

4.1.1 Position (LatLon)

A Position is a very small spot on the planet and its size depends on the precision. It is composed of an Y- and X-Ordinate, the Latitude and Longitude
It can be everywhere, on the poles or the Himalaya, places, which are not covered by the graph.

4.1.2 Vertex (Crossing)

Vertices are special Positions which are needed for the routing. Normal positions usually are nearby but not exactly on a vertex. But arbitrary positions are not directly routable, so we'll have to find the nearest Vertex within a reasonable distance instead.

4.1.3 POI (Point of Interest)

A POI is a well known location on the planet. It is equipped with a technical address (the index), a name and a permanent ID.
A POI usually has an associated Vertex but can also be a detached position.

4.2 Command families

Cargoloy's API is built upon four main command families, which reflect the recommended workflow. Usually a user starts with finding locations by name, then he or she needs some more information about the chosen points and finally requests a route between them.

4.2.1 lookup

This is the Geocoding interface.
It finds known POIs and other objects by name.

https://api.cargoloy.com/v1/lookup/pois/hamburg
https://api.cargoloy.com/v1/lookup/zones/seca

4.2.2 locate

This is the Reverse-Geocoding interface.
It returns detailed information of one location, which may be a POI, Position or Vertex.

POI by ID:	https://api.cargoloy.com/v1/locate/poi/@1
POI by idx	https://api.cargoloy.com/v1/locate/poi/1
Vertex by ID	https://api.cargoloy.com/v1/locate/vertex/1
Position	https://api.cargoloy.com/v1/locate/position/53.5/9.9

Have a deeper look into the response .
There is a **`locator`** attribute, pointing to either a **`position`**, **`poi`** or **`vertex`** child object. When **`locator=position`** then **`vertex`** and **`poi`** are usually **`nearby=true`**. But if a **`vertex`** has an associated **`poi`**, then the **`poi`** is **`nearby=false`**.
`locator=poi` gives us two options. Either a **`poi`** is connected to the network, then the corresponding **`vertex`** is **`nearby=false`** as well, or it is a detached position where a **`vertex`** can only be **`nearby=true`**.

4.2.3 route

This is the final request for calculating a route between two locations.

Position to Vertex	https://api.cargoloy.com/v1/route/53.5,9.9/2
Position to POI	https://api.cargoloy.com/v1/route/53.5,9.9/@2
POI to POI	https://api.cargoloy.com/v1/route/@1/@2

A route response is strongly structured and returns one leg per transportation type.
A typical route from a european to an american city needs three legs. We usually reach the first port by truck. To the second we go via sea and reach our destination on land again. These hops are reflected by the leg's source and target attributes. By convention the source of leg B is the same as the target of Leg A. Each leg is a selfcontained entity, which can be re-routed separately.

4.2.4 load

This family is meant for retrieving static system data.
Most of these methods are costly and should be called only once per session.
Ideally they are cached once on application level.

List of ports:	https://api.cargoloy.com/v1/load/ports
List of zones:	https://api.cargoloy.com/v1/load/zones
List of secas:	https://api.cargoloy.com/v1/load/secas
Default parameters:	https://api.cargoloy.com/v1/load/params

5 Alternative Routing

Routing results base on a default setup, giving each transportation type a more or less realistic weight. However, it is possible to change this behavior in order to prefer, avoid, delay or even block several objects like edges (e.g. streets), pois (e.g. ports) or entire regions (e.g. maritime straits, countries, etc.).

5.1 Parameter families

Deviant behavior, and thus alternative results, can be obtained by adding Query-Parameters to the URL. There are three major groups:

5.1.1 Penalties

Penalties prefer or avoid transportation types. Technically it is a simple multiplier, which increases or decreases the default costs of edges. Use values like 2 to double the costs or 10 and even higher to avoid types completely. Values between 0 and 1 decrease costs and hence prefer them. 0 itself means „use default“ and negative ones block completely.

- **`streetPenalty`** (trucks)
- **`riverPenalty`** (inland vessels)
- **`seaPenalty`** (seagoing ships)
- **`airPenalty`** (aircrafts)

The API also takes few maritime issues into account:

- **`secaPenalty`** (SECA and ECA zones)
- **`trafficPenalty`** (recommended maritime lanes)

5.1.2 Delays

Delays apply to hubs, such as ports or airports and add some kind of transshipment costs. This does not affect start points and destinations, but intermediate (via-) nodes on the route. The unit is one hour, so 24 means „wait for 1 day“. However, using delays in combination with other penalties might change the unit.

- **`viaPortDelay`**
- **`viaAirportDelay`**

5.1.3 Blockers

In contrast to penalties or delays where we can block all objects of one type by assigning a negative value, we should be able to block selected Objects as well. There are two different types ob objects, namely POIs and zones.

- **`blockPois`** (CSV of Poi-IDs to be blocked)
- **`blockZones`** (CSV of Zone-IDs to be blocked)

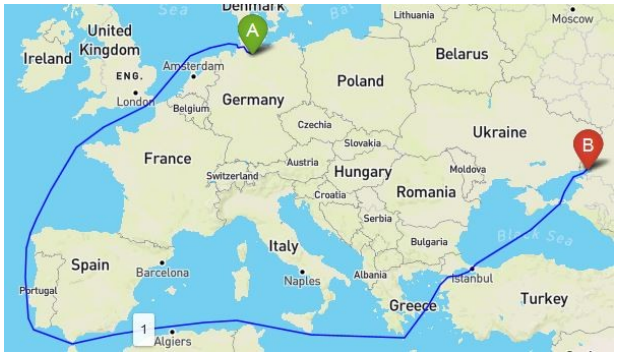
5.2 Examples

From Hamburg to Taganrog

https://api.cargoloy.com/v1/route/@1/@3249

```
params:
   trafficPenalty:    0
   seaPenalty:        0
   airPenalty:        -1
   streetPenalty:     20
   riverPenalty:      -1
   secaPenalty:       0
   viaPortDelay:      24
   viaAirportDelay:   24
   blockedZoneIds:    []
   blockedPoiIds:     []
```

(Snippet)



https://api.cargoloy.com/v1/route/@1/@3249?streetPenalty=0

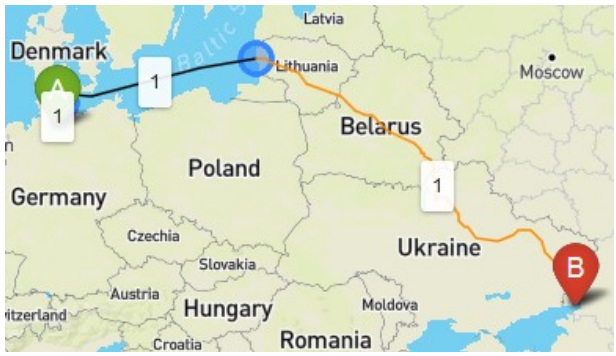
`streetPenalty=0`

Do not penalize streets with factor 20 and use the default setup instead

https://api.cargoloy.com/v1/route/@1/@3249?streetPenalty=0&seaPenalty=0.2&viaPortDelay=0

`streetPenalty=0`
`seaPenalty=0.2`
`viaPortDelay=0`

However, let's prefer the sea again and don't wait at ports.
And Ups! - We have just hopped over the Ports of Kiel and Klaipeda and got three legs.

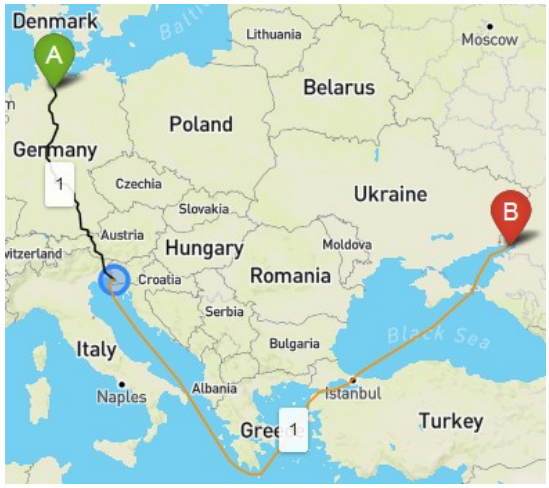


https://api.cargoloy.com/v1/route/@1/@3249?blockZones=2130&viaPortDelay=120

blockZones=2130
viaPortDelay=120

Back to defaults, but block the Strait of Gibraltar (Id 2130) and avoid hopping over too many ports by setting a delay of 5 days (120 h).

Now Trieste is our Via-Port.



https://api.cargoloy.com/v1/route/@1/@3249?blockZones=2130&viaPortDelay=120&blockPois=3283

If, for whatever reason, we do not like Trieste (Id 3283), we can block it as well by adding

blockPois=3283

Now Koper (Id 3284) is our next best candidate.
If Koper is not wanted either, just add it to the list:

blockPois=3283,3284