Linked Connections

Revolutionizing route planning frameworks:
publishing better data, building smarter clients.

Published: 2018-01-01
Editor: Pieter Colpaert - IDLab

Introduction

The Linked Connections 1.0 specification explains how to implement a data publishing server, and explains what you can rely on when writing a route planning client. Linked Connections uses "a connection" as its smallest building block: at all times, you will need to generate or process a departure stop, departure time, arrival stop and arrival time. These connections are then ordered by departure time, and split in fragments that do not exceed a reasonable page size for an HTTP response. Following the Linked Data principles and the REST constraints, we make sur ethat from any HTTP response, hypermedia controls can be followed to discover the rest of the dataset. A "Linked Connections graph" is a paged collection of connections, describing the time transit vehicles leave and arrive.

In this specification, we use 2 keywords. must is used when the client would not be able to use the data any longer. should is used in order to follow good practices for better performance, or to make the lives of developers easier.

Communication over HTTP

For each document that is published by a Linked Connections server, a Cross Origin Resource Sharing HTTP header must to be set as follows:

```
Access-Control-Allow-Origin: *
```

Next, the server should implement thorough caching strategies, as the cachability is one of the biggest advantages of the Linked Connections framework. Both conditional requests as regular caching are recommended.

Finally, as a generic data model, we use W3C's Resource Description Framework (RDF1.1). The server therefore must support at least one RDF1.1 format, such as JSON-LD, TriG or N-Quads. As a concequence, every connection will need to have a unique and persistent identifier: an HTTP URI.

This URI should follow the Linked Data principles.

A response's metadata

Each response from a Linked Connections page must contain some metadata about itself:

The URL after redirection, or the one indicated by the Location HTTP header, therefore must occur in the triples of the response. The current page must contain the hypermedia controls to discover under what conditions the data can be legally reused, and must contain the hypermedia controls to understand how to navigate through the paged collection(s).

There are different ways to retrieve the *legal disclaimer*. One option is to annotate each page with a dct:rights and dct:license property, linking each resource with its legal license and rights statement. This allows for fine-grained digital rights management, indicating for each document the legal conndditions before it may be reused. Another option is to link the document with a description of a dcat:Dataset through the property dcat:dataset (mind the capitals).

Also different ways exist to implement a paging strategy. At least one of the strategies must be implemented for a Linked Connections client to find the next pages to be processed.

1. Each response must contain a hydra:next and hydra:previous page link.
2. Each response should contain a hydra:search describing that you can search for a specific time, and that the client will be redirected to a page containing information about that timestamp. The hydra:property lc:departureTimeQuery is used to describe this functionality. For example:


```
{
  "@id": "https://graph.irail.be/sncb/connections?departureTime=2017-12-22T14:00:00.000Z",
  "@type": "hydra:PagedCollection",
  "hydra:next": "https://graph.irail.be/sncb/connections?departureTime=2017-12-22T14:10:00.000Z",
  "hydra:previous": "https://graph.irail.be/sncb/connections?departureTime=2017-12-22T13:50:00.000Z",   "hydra:search": {
    "@type": "hydra:IriTemplate",
    "hydra:template": "https://graph.irail.be/sncb/connections/{?departureTime}",
    "hydra:variableRepresentation": "hydra:BasicRepresentation",
    "hydra:mapping": {
      "@type": "IriTemplateMapping",
      "hydra:variable": "departureTime",
      "hydra:required": true,
      "hydra:property": "lc:departureTimeQuery"
    }
  }
}
```


Each response must support a Memento RFC gateway to ask for an archived version of the response. Clients can use this to make sure that stale responses are retrieved when planning routes over multiple pages, making sure no connections are skipped.

Describing connections

We want to be able to describe the basics of a connection, as well as use different properties.

The Linked Connections vocabulary

The Linked Connections vocabulary can be found at http://semweb.mmlab.be/ns/linkedconnections# (lc: from now on). It describes the basic properties to be used with the lc:Connection class:

lc:departureTime
: must Date-Time (including delay) at which the vehicle will leave for the lc:arrivalStop.

lc:departureStop
: must A gtfs:Stop

lc:departureDelay
: When the lc:departureTime is not the planned departureTime, this property must set the delay in seconds

lc:arrivalTime
: must Date-Time (including delay) at which the vehicle will arrives at lc:arrivalStop.

lc:arrivalStop
: must A gtfs:Stop

lc:arrivalDelay
: When the lc:arrivalTime is not the planned arrivalTime, this property must set the delay in seconds

The Linked GTFS vocabulary

GTFS is the de-facto standard today to exchange public transport data on the Web, and the terms used are familiar with transit app developers. We reuse these terms in the Linked GTFS vocabulary available at http://vocab.gtfs.org/terms# (further on: gtfs:), in order to still describe some much needed properties.

gtfs:trip
: Must be set to link an lc:Connection with a gtfs:Trip, identifying whether another connection is part of the current trip of a vehicle.

gtfs:pickupType
: Should be set to indicate whether people can be picked up at this stop. Possible values: gtfs:Regular, gtfs:NotAvailable, gtfs:MustPhone and gtfs:MustCoordinateWithDriver.

gtfs:dropOffType
: Should be set to indicate whether people can be dropped off at this stop. Possible values: gtfs:Regular, gtfs:NotAvailable, gtfs:MustPhone and gtfs:MustCoordinateWithDriver.

Example

An example in JSON-LD:

```
{
  "@id": "http://irail.be/connections/8863008/20171219/P8671",
  "@type": "Connection",
  "departureStop": "http://irail.be/stations/NMBS/008863008",
  "arrivalStop": "http://irail.be/stations/NMBS/008863354",
  "departureTime": "2017-12-19T15:50:00.000Z",
  "departureDelay": 60,
  "arrivalTime": "2017-12-19T16:20:00.000Z",
  "arrivalDelay": 60,
  "gtfs:trip": "http://irail.be/vehicle/P8671/20171219",
  "gtfs:pickupType":"gtfs:Regular",     "gtfs:dropOffType":"gtfs:Regular"
}
{
  "@context": {
     "lc": "http://semweb.mmlab.be/ns/linkedconnections#",
     "gtfs":"http://vocab.gtfs.org/terms#",
```

Addenda

```
      "xsd":"http://www.w3.org/2001/XMLSchema#",
      "Connection" : "lc:Connection",
      "departureStop": { "@id": "lc:departureStop", "@type":"@id"},
```

Default JSON-LD context used in this specification

```
      "arrivalStop": { "@id": "lc:arrivalStop", "@type":"@id"},
      "departureTime": {"@id":"lc:departureTime", "@type":"xsd:datetime"},
      "arrivalTime": {"@id":"lc:arrivalTime", "@type":"xsd:datetime"},
      "departureDelay": {"@id":"lc:departureDelay", "@type": "xsd:integer"},
      "arrivalDelay": {"@id":"lc:arrivalDelay", "@type": "xsd:integer"},
      "gtfs:headsign": {"@type":"xsd:string"},
      "gtfs:trip": {"@type": "@id"},
      "gtfs:route": {"@type": "@id"},
      "gtfs:pickupType":{"@type": "@id"},
      "gtfs:dropOffType": {"@type": "@id"}
    }
}
```