# A brief introduction to linked data

Joep Meindertsma, (https://twitter.com/@joepmeindertsma)03 Jul 2018

Linked data is a way to structure and share information, using links. These links make data more meaningful and useful. To understand why, let's take a piece of information and upgrade its data quality step by step, until it's linked data. In the later paragraphs, I'll get a little more technical. I'll discuss the RDF data model, serialization formats, ontologies and publishing strategies.

## Human Language

```
Tim is born in London on the 8th of June, 1955.
```

Humans understand what this sentence means, but to a computer, this is just a string of characters. If we wanted an application to do something with this sentence, such as display Tim's birthdate, we'd need the computer to understand English. A simpler solution would be to structure our information in a way that's useful to a computer.

## Tables

If we put the information in a table, we can simply let the computer read the `birthDate` field for Tim.

| name | birthPlace | birthDate |
|---|---|---|
| Tim | London | 06-08-1955 |

Great! By structuring data, computers can be programmed to do more useful things with it.

But now someone else wants to use this data and has a couple of questions.

- Who is Tim?
- Which London do you mean, the big one in the UK or the smaller one in Canada?
- Does `06-08` mean June 8th or August 6th?

## Links

Now, let’s add links to our data:

| name | birthPlace (http://schema.org/birthPlace) | birthDate (http://schema.org |
|---|---|---|
| Tim (https://www.w3.org/People/Berners-Lee/) | London (http://dbpedia.org/resource/London) | 1955-06-08 |

By adding these links, others can answer all previous questions by themselves. The links solve three problems:

- **Links provide extra information.** Follow the link to Tim to find out more about him.
- **Links remove ambiguity.** We now know exactly which London we’re talking about.
- **Links add standardization.** The birthDate link tells us we need to use the YYYY-MM-DD notation.

These three characteristics make linked data more reusable. The data quality has been improved, because other people and machines can now interpret and use the information more reliably.

Let’s take a step back and look at what went wrong in our first non-linked table. The problems were obvious to someone who reuses the data but were not that relevant for the creator of the table. I made the table, I knew which Tim and London I was talking about, I knew how the birthdate should be read. This closed worldview is the root cause of much of the problems in digital systems today. Developers tend to make software that produces data that only they can fully understand. They have their own assumptions, identifiers, and models. Linked data solves this problem by creating *consensus* about what data represents and how it should be interpreted.

## Triples & the RDF data model

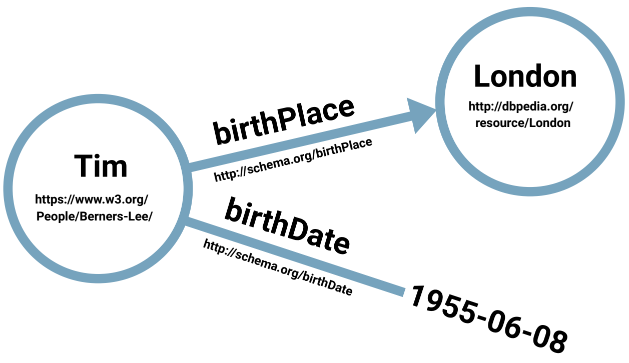
In the example above, we’re making two separate statements about Tim: one about his birthdate and one about his birthplace. Each statement had it’s own cell in the table. In linked data, these statements are called *triples*. That’s because every triple statement has three parts: a *subject*, a *predicate*, and an *object*.

| Subject | Predicate | Object |
|---|---|---|
| Tim (https://www.w3.org/People/Berners-Lee/) | birthPlace (http://schema.org/birthPlace) | London (http://dbpedia.org/resou |

| Subject | Predicate | Object |
|---|---|---|
| Tim (https://www.w3.org/People/Berners-Lee/) | birthDate (http://schema.org/birthDate) | 1955-06-08 |

A bunch of triples about a single subject (such as Tim) is called a *resource*. That's why we call this data model the Resource Description Framework: *RDF*. It is the de facto standard for linked data.

Instead of using a table of triples, we could visualize the RDF data as a *graph* (https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)).


London
http://dbpedia.org/resource/London
birthPlace
http://schema.org/birthPlace
Tim
https://www.w3.org/People/Berners-Lee/
birthDate
http://schema.org/birthDate
1955-06-08

The object of the first triple, for the birthPlace, contains a link (an *IRI* (https://en.wikipedia.org/wiki/Internationalized_Resource_Identifier)) to some other resource (London). The object of the second triple (the birthDate) is not a link, but a so-called *literal value*. The literal value cannot have any properties, since it's not a resource.

That's a lot of concepts that can be a bit confusing at first. However, they will appear all the time when you're actually working with linked data, so try to get an accurate mental model of these concepts.

Again, let's take a step back and reflect. What can we say about the RDF model, looking at how it works? First, this shows that RDF is actually a ridiculously *simple* model. You can represent anything in RDF with just three columns. Second, you should note that it is not possible to add extra information on edges (these arrows in the graph). This is different from most graph models, where edges can have their own properties. Another characteristic of the RDF model is that it is really easy to combine two RDF graphs. Integrating two datasets is a luxury that most data models don't have. Finally, having a database model that is decoupled from your application models, means high *extensibility* and *flexibility*. Changing your model or adding properties do not require any schema changes. This makes RDF so great for systems that change over time.

## RDF Serialization

Let’s get a little more technical (feel free to skip to Ontologies if you don’t like all this code). RDF is a *data model*, not a *serialization format*. In other words: The subject, predicate, object model can be represented in several ways. For example, here’s the same triples from the table and the graph above, serialized in the *Turtle* format:

Turtle
```
<https://www.w3.org/People/Berners-Lee/>
<http://schema.org/birthDate> "1955-06-08".
<https://www.w3.org/People/Berners-Lee/>
<http://schema.org/birthPlace>
<http://dbpedia.org/resource/London>.
```

The <> symbols indicate IRIs and the `""` symbols indicate literal values.

This example doesn’t look as good as the graph above, right? Long URLs tend to take up a lot of space and make the data a bit tough to read. We can use *namespaces* (denoted with `@prefix`) to compress RDF data and make it more readable.

Turtle
```
@prefix tim: <https://www.w3.org/People/Berners-Lee/>.
@prefix schema: <http://schema.org/>.
@prefix dbpedia: <http://dbpedia.org/resource/>.

<tim> schema:birthDate "1955-06-08".
<tim> schema:birthPlace <dbpedia:London>.
```

You could also express the same RDF triples as JSON-LD:

JSON
```
{
  "@context": {
    "schema": "http://schema.org/",
    "dbpedia": "http://dbpedia.org/resource/"
  },
  "@id": "https://www.w3.org/People/Berners-Lee/",
  "schema:birthDate": "1955-06-08",
  "schema:birthPlace": {
    "@id": "dbpedia:London"
  }
}
```

Or as HTML with some extra RDFa attributes:

HTML
```
<div xmlns="http://www.w3.org/1999/xhtml"
  prefix="
    schema: http://schema.org/
    rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
    rdfs: http://www.w3.org/2000/01/rdf-schema#"
  >
  <p typeof="rdfs:Resource"
about="https://www.w3.org/People/Berners-Lee/">
    Tim
    <span rel="schema:birthPlace"
resource="http://dbpedia.org/resource/London">
      is born in London
    </span>
    <span property="schema:birthDate" content="1955-06-08">
      on the 8th of June, 1955
    </span>
  </p>
</div>
```

The Turtle, JSON-LD and HTML+RDFa each contain the same RDF triples and can be automatically converted into each other. You can try this for yourself (http://rdf-translator.appspot.com/) and discover even more RDF serialization formats, such as microformats, RDF/XML (don't use this, please (https://github.com/mhausenblas/rdfxml.info/blob/master/input/RDF-XML%20sucks%20-%20praise%20and%20damnation.txt)) and N-Triples.

The number of serialization options for RDF might be a bit intimidating, but you shouldn't feel the need to understand and know every single one. The important thing to remember is that there's a lot of options that are compatible with each other and use the RDF data model.

## Ontologies

Let's tell a bit more about Tim. First of all, it might be useful to specify that Tim is a person:

Turtle
```
@prefix tim: <https://www.w3.org/People/Berners-Lee/>.
@prefix schema: <http://schema.org/>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix foaf: <http://xmlns.com/foaf/spec/#term_>.

<tim> a <foaf:Person>;
  schema:birthDate "1955-06-08";
  schema:birthPlace <dbpedia:London>.
```

We've referred to `foaf:Person` (http://xmlns.com/foaf/spec/) to specify that Tim is an *instance* of the class *Person*. Foaf (Friend Of A Friend) is an *ontology* that is designed to describe data related to people in social networks. It defines the concept of Person and some attributes, such as a profile image. We used the `schema.org` ontology for the concepts of `birthDate` and `birthPlace`.

There exist many ontologies, ranging from organizations (https://www.w3.org/TR/vocab-org/) (which describes concepts like *memberships*) to pizza (https://protege.stanford.edu/ontologies/pizza/pizza.owl) (which describes concepts like *ingredients*). These ontologies are often described in the OWL format, which is a subset of RDF.

That means that OWL ontologies are open data models that can be interpreted by machines.

Having an machine-readable data model opens up some really cool possibilities. You can generate documentation (https://github.com/dgarijo/Widoco). You can use reasoners to *infer* new knowledge about your data. You can even generate forms and other UI components in React using Link-Lib (https://github.com/fletcher91/link-lib).

The power of the ontology goes far, but that deserves its own article.

## Publishing linked data

Linked data is meant to be shared. We can do this in several ways:

Firstly, there's the **data dump**. Serialize your RDF the way you like and make it accessible as a single file. It's the easiest and often the cheapest way to publish your data. However, if someone just wants to know something about a single subject (or resource) in your data dump, he'd have to download the entire data dump. That's cumbersome, and makes your data not as re-usable as it could be. All processing and querying efforts are left to the downloader. Furthermore, data dumps are hard to manage and therefore likely to be outdated.

**Subject pages** to the rescue! Make the RDF data available through HTTP at the location where you'd expect it: at *the same link as the resource IRI*. Doing this makes your data truly linked, since every resource can now be downloaded separately and automatically. Subject pages can be either *static* or *dynamic*. Static subject pages are simply RDF files hosted on some URL. Sharing static subject pages is very simple, but static data is hard to maintain or edit. Dynamic pages are generated by a server, so the underlying data could be edited by any framework. Another advantage of using dynamic subject pages, is that you can serialize to many different formats. You can show HTML to humans and RDF to computers. For example, our project Argu (https://argu.co) (an online democracy and discussion tool) works like this. Visit a webpage (or subject page) (e.g. argu.co/nederland/m/46 (https://argu.co/nederland/m/46)). If you want the same content as linked data, add a serialization extension (e.g. .ttl (https://argu.co/nederland/m/46.ttl)) or use HTTP Accept Headers (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept). Note that even though this project serializes to all sorts of RDF formats, the project itself does not use an RDF database / triple store.

Perhaps the most popular and easiest way to publish linked data is with **annotated pages**. Remember the RDFa serialization format, discussed above? That's annotated pages. Using RDFa or Microdata in your existing web pages provides some benefits, especially to SEO. For example, you can get these cool boxes in google (https://developers.google.com/search/docs/guides/intro-structured-data?visit_id=1-636649250129274125-3755783713&hl=en&rd=1), which show things like star ratings in search previews. However, annotated pages are more for adding a bit of spice to your existing webpage than to make huge datasets available. Parsing (reading) RDFa from a large HTML document will always be more expensive than reading Turtle or any other simple triple RDF format.

A radically different way to share your linked data is through a **SPARQL** endpoint. SPARQL is a *query language*, like SQL, designed to perform complex search queries in large RDF graphs. With SPARQL, you can run queries such as 'which pianists live in the Netherlands', or 'what proteins are involved in signal transductions and related to pyramidal neurons?'. SPARQL is without any doupt extremely powerful, but using it as the primary measure to share your RDF data might be difficult for your project. If you want one, you will probably need to store your RDF data in a specialized triple store that features a SPARQL endpoint. Not many databases do, unfortunately, so you'll be limited to either proprietary solutions or projects with relatively little adoption and support. Ask yourself if your users will need to run complex queries on your data. For most linked data projects, I'd recommend to use a conventional database and serialize the data to some RDF format when a user sends a request, i.e. use the aforementioned subject pages pattern instead of a SPARQL endpoint.

Other technologies like Linked Data Fragments (http://linkeddatafragments.org/) and HDT (http://www.rdfhdt.org/what-is-hdt/) allow for even more efficient sharing and storing of linked data.

Note that there's a difference between linked data and linked *open* data. Although linked data would be a great choice for publishing open data, you don't have to make your linked data accessible to others. It's perfectly possible to secure linked data.

## Further reading

If you want to learn more about the vision behind the semantic web and linked data, read the 2006 paper (https://eprints.soton.ac.uk/262614/1/Semantic_Web_Revisted.pdf) by some of the original inventors. If you're looking for inspiration and example projects, check out the Linked Open Data Cloud (https://lod-cloud.net/). If you want to learn more about reasoning and ontologies, try the W3C OWL primer (https://www.w3.org/TR/2012/REC-owl2-primer-20121211/). For SPARQL, the Apache Jena tutorial (https://jena.apache.org/tutorials/sparql.html) could help.

And of course, if you want to get help with your linked data project, feel free to send me an email (mailto:joep@argu.co)!

Best practices for RESTful API design » (/blog/api-design/)

**Ontola**
Home (/)
About us (/about)
Blog (/blog)
Contact (/contact)
Nederlands (/nl/what-is-linked-data/)

**What we do**
Linked data (/)
Software development (/software-development)
Continuous integration (/continuous-integration)

**Find us**
(https://github.com/ontola) (https://www.linkedin.com/company/ontola/)
(https://www.facebook.com/ontola.io/)
(https://twitter.com/ontola_io)
info@ontola.io (mailto:info@ontola.io)
+316 360 209 42 (tel:0031636020942)
Koningslaan 62, Utrecht, NL (https://goo.gl/maps/x5Sggs9X7L32)
Argu B.V. (https://argu.co) kvk 65684168